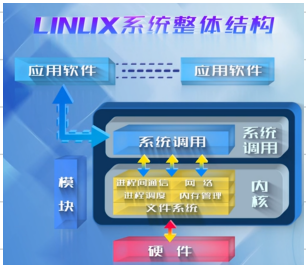
Linux

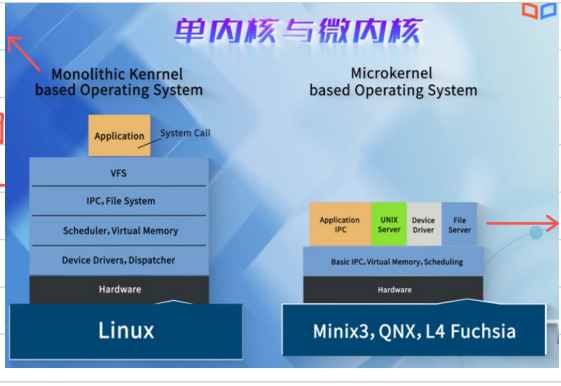
1.1 Linux 操作系统概述



→ 隔离变化

1.2 Linux 内核模块以及内核模块编程

3系统之间可以直接调用性能高但可维护性差。



→ 以服务器方式放在外部通信导致效率低，但可维护性强

Linux 目录结构

内核编程

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
/*
    模块的初始化函数lkp_init()
    __init是用于初始化的修饰符
*/
static int __init lkp_init(void)
{
    printk("<1>Hello,world!from the kernel space...\n");
    return 0;
}
/*
    模块的退出和清理函数lkp_exit()
*/
static void __exit lkp_exit(void)
{
    printk("<1>Goodbye,world!leaving kernel space...\n");
}

module_init(lkp_init);
module_exit(lkp_exit);
/*
    模块的许可证声明GPL
*/
MODULE_LICENSE("GPL");
```

module头文件

printk

模块入口

输出到日志文件

编译修饰符，模块执行后回收内存空间.

卸载模块时执行

LINUX内核模块编程入门

内核模块不是独立的可执行文件，但在运行时其目标文件被链接到内核中。只有超级用户才能加载和卸载模块。

```
obj-m:=module_example.o                    #产生module_example模块的目标文件
CURRENT_PATH := $(shell pwd)               #模块所在的当前路径
LINUX_KERNEL := $(shell uname -r)          #linux内核源代码的当前版本
LINUX_KERNEL_PATH := /usr/src/linux-headers-$(LINUX_KERNEL)
                                           #linux内核源代码的绝对路径
all:
    make -C $(LINUX_KERNEL_PATH) M=$(CURRENT_PATH) modules   #编译模块
clean:
    make -C $(LINUX_KERNEL_PATH) M=$(CURRENT_PATH) clean     #清理模块
```

目前ubuntu 不同系统makefile有不同的要求.

obj-m :=这个赋值语句的含义是说明要使用目标文件module_example.o建立一个模块，最后生成的模块名为module_example.ko。.o文件是经过编译和汇编，而没有经过链接的中间文件。

注：makefile文件中，若某一行是命令，则它必须以一个Tab键开头。

编译好之后即可插入内核.

LINUX内核模块编程入门

模块插入命令：
$insmod module_example.ko
模块删除命令：
$rmmod module_example

LINUX内核模块与C应用的对比

	C语言应用程序	内核模块程序
使用函数	Libc库	内核函数
运行空间	用户空间	内核空间
运行权限	普通用户	超级用户
入口函数	main()	module_init ()
出口函数	exit()	module_cleanup ()
编译	gcc -c	make
连接	gcc	insmod
运行	直接运行	insmod
调试	gdb	kgdb

1.3 Linux 内核源码中的双链表结构

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;
};
```

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。

· 链表的声明和初始化

内核代码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*/
```

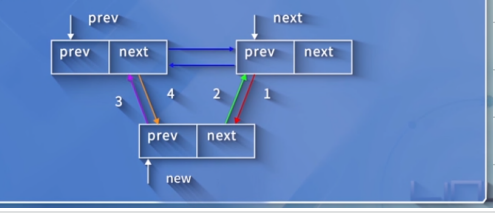
```
static inline int list_empty(const struct list_head *head)
{
return head->next == head;
}
```

新节点的插入

对函数作用域的一个限制，表示函数的作用域仅限于本文件（信息隐藏）

函数对编译程序是可见的

```
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;
}
```




```
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。

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

头尾相连，循环结束

Question：只能找到节点在列表中的偏移位置

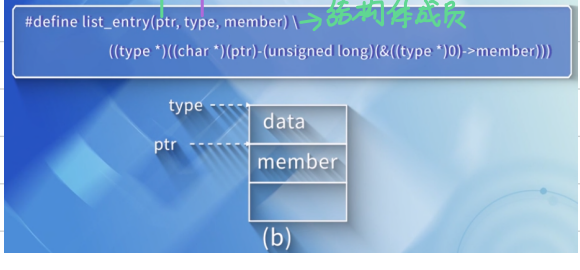
指针，member的在位置

结构体类型

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

结构体成员

→ 获得节点的起始地址.




```
(
    (type*)
    (
        ① (char*)(ptr) -
        ② (unsigned long) (&((type*)0)->member)
    )
)
```

① 指针绝对位置

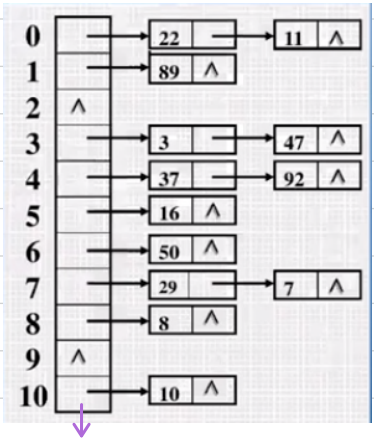
② 偏移量

得到起始位置.

1.4 源码分析 内核中的哈希表

哈希表：关键字与存储位置之间建立映射关系

哈希表冲突解决方法 ①开放寻址 ②再散列 ③链地址法



数组，链表的头结点

用于实现哈希表中的链地址法 .

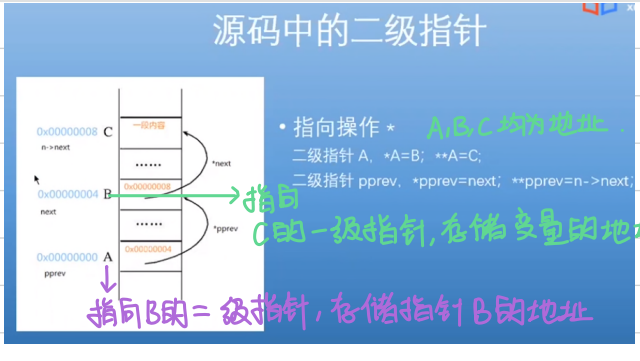
```
struct hlist_head {
    struct hlist_node *first;
};

struct hlist_node {
    struct hlist_node *next, **pprev;
};
```

→ 二级指针

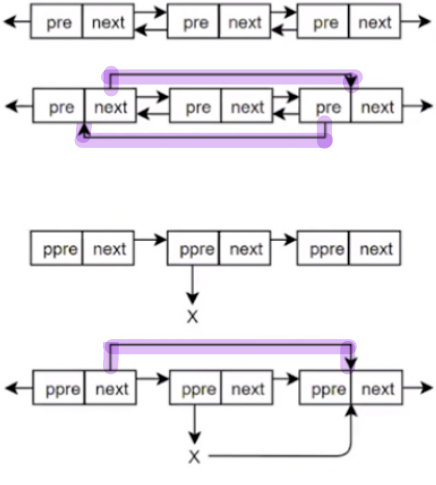
"→", "." 访问类的成员变量或成员函数

"." 用于普通变量操作

"→" 用于指针变量操作

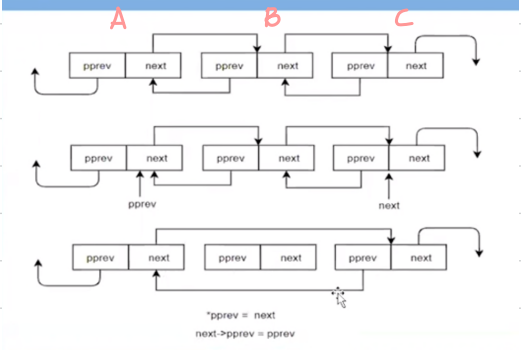


双链表中的删除




```
static __inline__ void __hlist_del(struct hlist_node *n)
{
    struct hlist_node *next = n->next;
    struct hlist_node **pprev = n->pprev;
    *pprev = next;
    if (next)
        next->pprev = pprev;
}
```

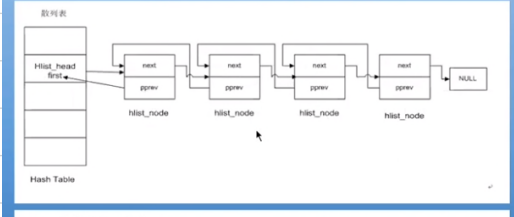
在哈希表中删除一个节点



*ppre = next

next->ppre = ppre

内核中的哈希表




```
struct hlist_head {
    struct hlist_node *first;
};

struct hlist_node {
    struct hlist_node *next, **pprev;
};
```

1.5 动手实践 Linux 内核模块的插入和删除

```
zcy@ubuntu: ~/Desktop/Mooc
File  Edit  View  Search  Terminal  Help
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>

static int __init lkm_init(void)
{
    printk("Hello World\n");
    return 0;
}

static void __exit lkm_exit(void)
{
    printk("Goodbye\n");
}

module_init(lkm_init);
module_exit(lkm_exit);

MODULE_LICENSE("GPL");
~
~
~
~
-- INSERT --
```

支持模块机制

入口函数

支持消息打印级别

出口函数

入口函数

出口函数

声明许可证

Makefile 文件

```
zcy@ubuntu: ~/Desktop/Mooc
File  Edit  View  Search  Terminal  Help
obj-m:=helloworld.o

CURRENT_PATH:=$(shell pwd)
LINUX_KERNEL:=$(shell uname -r)
LINUX_KERNEL_PATH:=/usr/src/linux-headers-$(LINUX_KERNEL)

all:
    make -C $(LINUX_KERNEL_PATH) M=$(CURRENT_PATH) modules
clean:
    make -C $(LINUX_KERNEL_PATH) M=$(CURRENT_PATH) clean
~
```

目标文件

内核代码版本

内核路径

Tab

```
zcy@ubuntu:~/Desktop/Mooc$ vim helloworld.c
zcy@ubuntu:~/Desktop/Mooc$ ls
helloworld.c   helloworld.mod.c  helloworld.o  modules.order
helloworld.ko  helloworld.mod.o  Makefile      Module.symvers
zcy@ubuntu:~/Desktop/Mooc$ sudo insmod helloworld.ko
[sudo] password for zcy:
zcy@ubuntu:~/Desktop/Mooc$ lsmod
Module                  Size  Used by
helloworld I           16384  0
rfcomm                 77824  4
bnep                   20480  2
crct10dif_pclmul       16384  0
```

模块已在