操作系统课程设计

一. 目的

- 掌握Linux操作系统的使用方法；
- 了解Linux系统内核代码结构；
- 掌握实例操作系统的实现方法。

知识点：

✓用户界面：系统调用与图形界面；

✓内核实现：内核生成；
字符驱动实现；
文件系统。

二. 设计内容

1. Linux用户界面的使用

- 编一个C程序，其内容为实现文件拷贝的功能。
  基本要求：使用系统调用open/read/write...；
  选择：容错、cp。

- 编一个C程序，其内容为分窗口同时显示三个并发进程的运行结果。要求用到Linux下的图形库。 (gtk/Qt)

基本要求：三个独立子进程，各自窗口显示；
*选择：*三个进程誊抄演示。

2. 掌握系统调用的实现过程，通过编译内核方法，增加一个新的系统调用。另编写一个应用程序，使用新增加的系统调用。

(1) 内核编译、生成，用新内核启动；

(2) 新增系统调用实现：文件拷贝或P、V操作。

3. 掌握增加设备驱动程序的方法。通过模块方法，增加一个新的字符设备驱动程序，其功能可以简单,基于内核缓冲区。

基本要求：演示实现字符设备读、写；

选择：键盘缓冲区，不同进程、追加、读取。

4.使用GTK/QT实现系统监控器

(⑴ 了解/proc文件的特点和使用方法;

(⑵ 监控系统中进程运行情况;

(⑶ 用图形界面实现系统资源的监控。

5．设计并实现一个模拟的文件系统

(1)基于一大文件(10M或100M)，模拟磁盘;

(2)格式化，建立文件系统管理数据结构;

(2)基本操作，实现文件、目录相关操作。

三. 设计说明

1. 实验环境

- windows +虚拟机linux （30G+）或单独Linux分区
- linux系统版本
  - Fedora 5.0 6.0
  - ubuntu 14.04 16.04 18.10 ...
  - 内核版本 linux-2.6 Linux-2.31 Linux-3.18 Linux-4.x ...

Linux根文件系统目录结构

- /boot: Linux内核启动相关文件
- /dev:dev是device（设备）的缩写。这个目录下是所有Linux的外部设备，在Linux中设备和文件是用同种方法访问的。例如：/dev/hda代表第一个物理IDE硬盘;
- /etc:这个目录用来存放系统管理所需要的配置文件和子目录;
- /lib:这个目录里存放着系统最基本的动态链接共享库，/lib/modules 内核对应的可加载模块

- /usr: 这是最庞大的目录，我们要用到的应用程序和文件几乎都存放在这个目录下。其中包含以下子目录:
  - /usr/include:Linux下开发和编译应用程序需要的头文件，在这里查找;
  - /usr/lib:存放一些常用的动态链接共享库和静态档案库;
  - /usr/local:这是提供给一般用户的目录，在这里安装软件最适合;
  - /usr/man:帮助文档的存放目录;
  - /usr/src: 由rpm安装的Linux开放的源代码就存在这个目录。

Linux内核源代码的文件组织

Linux核心源代码位于
http://www.kernel.org/

编号约定：任何偶数的核心（例如2.4.7）都是一个稳定的发行的核心，而任何奇数的核心（例如2.1.42）都是一个开发中的核心。

- 核心源程序的文件按树形结构进行组织，简要介绍目录结构如下：

- **arch**:arch子目录包括了所有和体系结构相关的核心代码。它的每一个子目录都代表一种支持的体系结构，例如i386就是关于intel cpu及与之相兼容体系结构的子目录，PC机一般都基于此目录;
- **drivers**:放置系统所有的设备驱动程序;每种驱动程序又各占用一个子目录，如/block下为块设备驱动程序;
- **include**:include子目录包括编译核心所需要的大部分头文件。与平台无关的头文件在include/linux子目录下，与 intel cpu相关的头文件在include/asm-i386子目录下;

- **init**:这个目录包含核心的初始化代码(注：不是系统的引导代码)，包含两个文件 main.c和 version.c，这是研究核心如何工作的一个非常好的起点；
- **mm**:这个目录包括所有独立于 cpu 体系结构的内存管理代码，如页式存储管理内存的分配和释放等，而和体系结构相关的内存管理代码则位于arch/*/mm/；
- **kernel**:主要的核心代码，此目录下的文件实现了大多数Linux系统的内核函数，其中最重要的文件当属进程调度sched.c，同样，和体系结构相关的代码在arch/*/kernel中。

2. Linux编程环境

1、函数库

glibc :要构架一个开发环境，glibc是必不可少的，它是Linux下C的主要函数库。

2、编译器

gcc(GNU CCompiler)是GNU推出的功能强大、性能优越的多平台编译器，gcc编译器能将C、C++语言源程序、汇编程序和目标程序编译、连接成可执行文件.

3、系统头文件

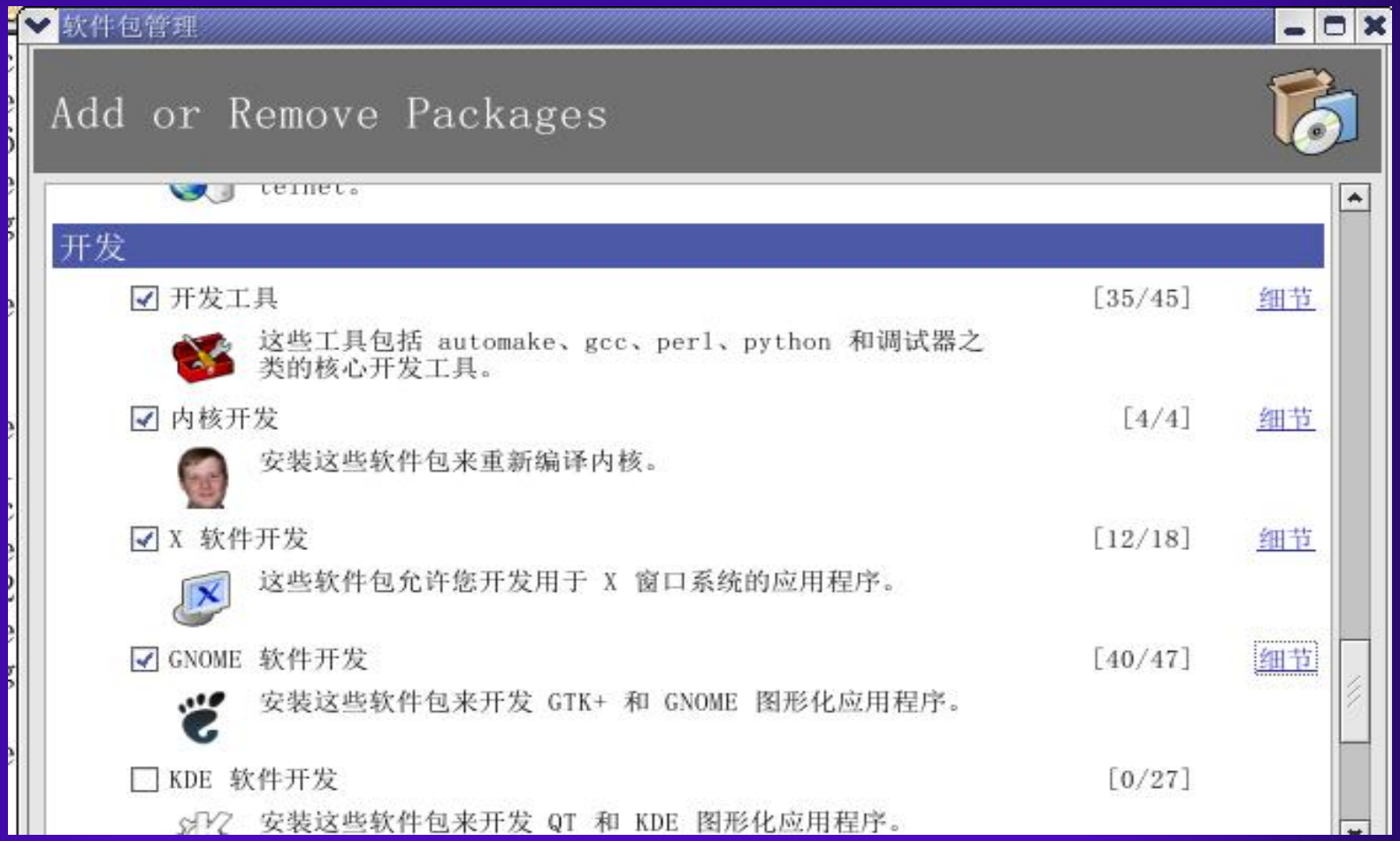
glibc_header

缺少了系统头文件 ,就会无法编译C源程序

4、其他软件: vi，rpm , tar，binutils, make

5、开发环境相关软件包的下载

Fedora 软件包管理:system-config-packges

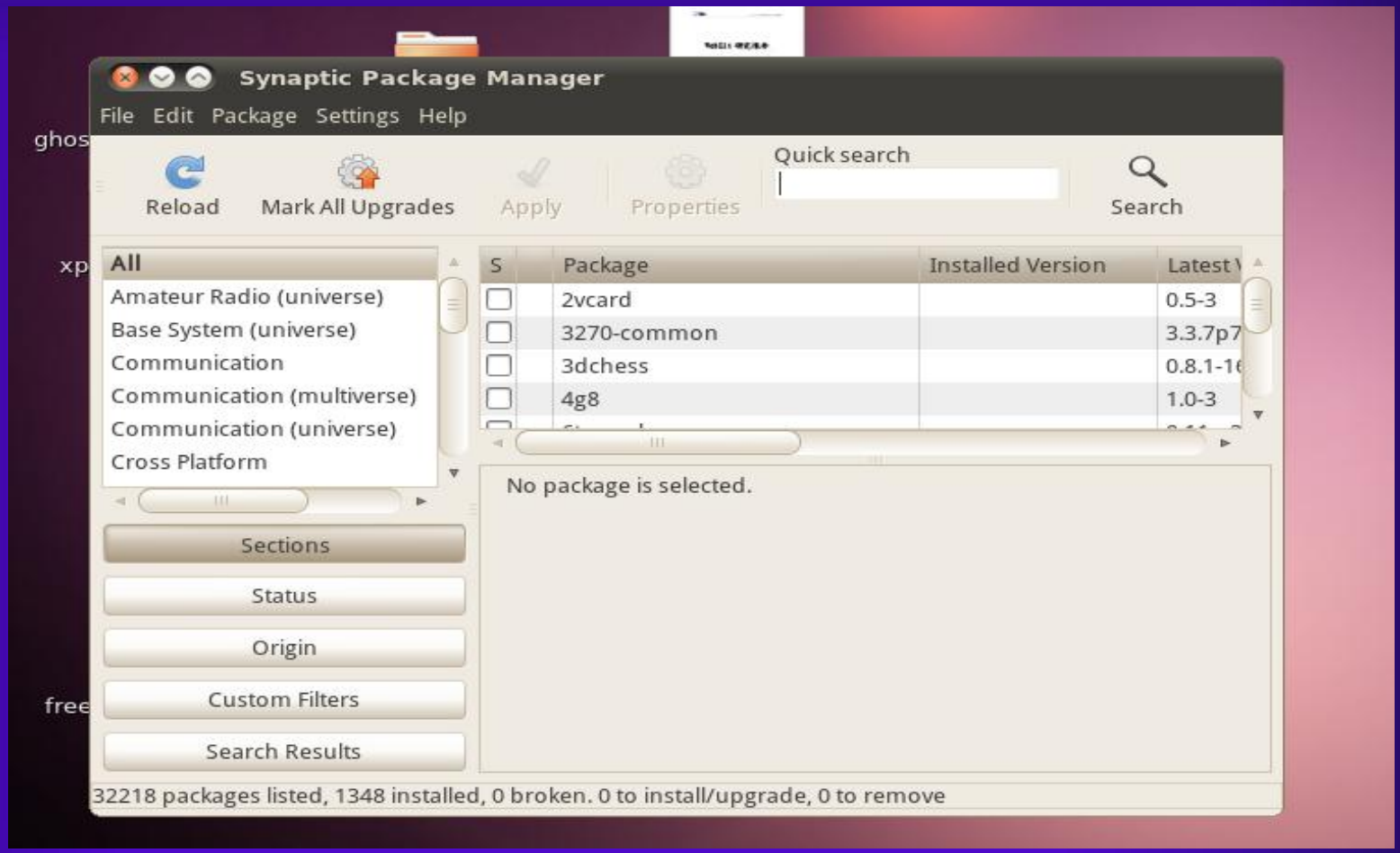


程序调试

gdb

gcc –g

Ubuntu 软件包管理



$sudo apt-get install package

$apt-get install libncurses5-dev

Linux的图形编程

X-Window已经成为了Unix/Linux图形界面的标准。 MIT 1984

X-Window分为三个层次：

1） X-SERVER，与显卡相关的中间层

服务程序用来控制实际的显示设备和输入设备(键盘和鼠标或其他输入设备)根据客户程序请求建立窗口、在窗口中画图形、图像和文字；

2） X-Window底层库，是最低实现层(XLib)；

3） 窗口管理器，实现最终用户界面：最常见的是KDE（K Desktop Environment）和GNOME（GNU Network Object Model Environment）。

X-Window基于一种客户机/服务器的思想。应用程序可以使用底层的Xlib API、也可以使用高级一些的QT、GTK + 等图形用户界面库。

- gtk

http://www.gtk.org

具有oo特色的C语言框架

- qt

http://www.trolltech.com

- ✓ C++ 图形用户界面库
- ✓ 跨平台
- ✓ 面向对象，模块化程度非常高，可重用性较好
- ✓ 有丰富的 API和控件
  支持 2D/3D 图形渲染，支持 OpenGL

- 编写gtk程序

1 初始化Gtk

2 建立控件

3 登记消息与消息处理函数

4 执行消息循环函数gtk_main()

只有gtk_main_quit()函数才能停止Gtk+的执行，从而最终退出应用程序。把gtk_main_quit()函数放在某个消息处理函数之中

- 编译和执行

程序中用到Gtk+函数或定义的每一部分必须包含gtk/gtk.h文件，此外，还必须连接若干库。

gcc hello.c -o hello `pkg-config --cflags``pkg-config --libs` 反引号（在键盘上位于字符1的左边

chmod -777 hello”将hello设定为可执行的文件。

3. Linux系统调用

- Linux内核中设置了一组用于实现各种系统功能的子程序，称为系统调用。用户可以通过系统调用命令在自己的应用程序中调用它们。

系统调用　核心态　操作系统核心提供

普通的函数调用　用户态　函数库或用户自己提供

很多已经被我们习以为常的C语言标准函数，在Linux平台上的实现都是靠系统调用完成的，如 open(), close(), malloc(), fork() .所以如果想对系统的原理作深入的了解，掌握各种系统调用是初步的要求。

系统调用工作原理

不能访问内核所占内存空间也不能调用内核函数。进程调用一个特殊的指令，这个指令会跳到一个事先定义的内核中的一个位置（当然，这个位置是用户进程可读但是不可写的）。在Intel CPU中，这个由中断INT0x80实现。跳转到的内核位置叫做sysem_call。检查系统调用号，这个号码代表进程请求哪种服务。然后，它查看系统调用表(sys_call_table)找到所调用的内核函数入口地址。接着，就调用函数，等返回后，做一些系统检查，最后返回到进程（或到其他进程，如果这个进程时间用尽）。系统调用号表示数组sys_call_table[]中的位置。

Linux2.6.6:sys_call_table在arch/i386/kernel/entry.S中：

Entry(system_call_table)

Linux2.6.31:sys_call_table在文件arch/x86/kernel/syscall_table_32.S中

如何使用系统调用

```
#include<linux/unistd.h>  /*定义宏_syscall1*/
 #include< time.h > /*定义类型time_t*/
_syscall1(time_t,time,time_t *,tloc) /*宏，展开后得到time()函数的原型*/
main(){
time_t the_time;
the_time=time((time_t *)0); /*调用time系统调用*/
printf("The time is %ld\n",the_time);
}
```

标准的系统调用的形式，宏_syscall1()展开来得到一个函数原型，如果把程序改成下面的样子，程序也可以运行得同样的结果。

```
#include< time.h >
main(){
time_t the_time;
the_time=time((time_t *)0); /*调用time系统调用*/
printf("The time is %ld\n",the_time);
}
```

time.h中已经用库函数的形式实现了time这个系统调用，省掉了调用_syscall1宏

大多数系统调用都在各种C语言函数库中有所实现，一般情况下，都调用普通的库函数

如何使用系统调用续

- 自2.6.19版本开始，_syscall宏被废除，我们需要使用syscall函数，通过指定系统调用号和一组参数来调用系统调用。
- syscall函数原型为：其中number是系统调用号，number后面应顺序接上该系统调用的所有参数。

增加系统调用注意事项：

- mm_segment_t old_fs =get_fs()
- set_fs (KERNEL_DS) //放过用户态参数检查
- set_fs(old_fs)

增加一个新的系统调用

- 实现方式

- 把新的系统调用永久性的加入内核中，需要编译生成新的内核；
- 以模块的方式加入系统调用，可在运行时添加，不须重新编译内核。

要求：采用编译内核方式！

Fedora/Ubuntu 10.04+linux 2.6+grub 1.0

编译内核---生成核心映像

#bzip2 –d linux-2.6.6.tar.bz2

#tar　xvf　linux-2.6.6.tar

#cd　/linux-2.6.6

#make menuconfig

存盘退出，生成内核配置文件

#make bzImage

#cp /linux-2.6.6/arch/i386/boot/bzImage
　/boot/vmlinuz-2.6.6

编译内核---生成对应核心模块

#make modules

#make modules_install

modules安装到/lib/modules/2.6.6

#mkinitramfs -o /boot/initrd-2.6.6.img 2.6.6

修改 /boot/grub/grub.conf如下：

```
default=0
timeout=50
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
hiddenmenu
title Fedora Core (2.6.9-1.667)
root (hd0,0)
kernel /boot/vmlinuz-2.6.9-1.667 ro
    root=LABEL=/ rhgb quiet
initrd /boot/initrd-2.6.9-1.667.img
```

/*类似增加自己的核心*/
title Fedora Core (2.6.6)
    root (hd0,0)
    kernel /boot/vmlinuz-2.6.6 ro root=LABEL=/ rhgb quiet
initrd /boot/initrd-2.6.6.img

Ubuntu 14.04+linux 3.xx+grub 2

- make menuconfig　//生存配置文件.config
- make bzImage　//生存内核压缩映像
- make modules　//生存模块
- make modules_install
  //安装模块到/lib/modules/版本
- make install　//安装内核并修改grub

注:

linux 3.18　linux 3.8　linux4.4 linux4.8 成功;

linux 3.11　　图形界面登录失败;

- 直接修改/etc/default/grub(然后update-grub)

GRUB_HIDDEN_TIMEOUT=0

- 此配置将影响菜单显示。若设置此选项,将在此时间内隐藏菜单而显示引导画面。
- 菜单将会被隐藏,除非在此行开头加上一个 # 符号。(# GRUB_HIDDEN_TIMEOUT=0)。
  若是大于 0 的整数,系统将会依此配置的秒数暂停,但不会显示菜单。 0 则菜单不会显示,也不会有延迟。
- 使用者可以在启动时按住 SHIFT 键不放以强制显示菜单。
- 启动过程中,系统将会检查 SHIFT 键状态。若无法辨识按键状态,会有一个短时间的延迟让使用者可通过按下 ESC 键来显示菜单。

GRUB__TIMEOUT=10 菜单显示时间

增加系统调用步骤

- 修改系统调用表，如在文件arch/i386/kernel/entry.S中添加:（系统调用表跟内核版本以及32/64位相关）

  . long SYMBOL_NAME(sys_mySysCall)()

- 定义系统调用号，

  在文件/linux-2.6.6/include/asm/unistd.h中添加:

  #define __NR_mySysCall　2xx

  现在高版本内核一般不再需要定义系统调用号。

- 添加自己的代码，修改/kernel/sys.c:

  asmlinkage int sys_mySysCall(arg1 , arg2…)

  {

  ……

  }

- 重新编译内核，进入新内核，编写测试程序

```
asmlinkage int sys_mysyscall(struct timeval * tv)
```

test.c

```
#include </linux-2.6.6/include/asm/unistd.h>
/* 根据参数个数，使用_syscall0到_syscall6来定义*/
 _ syscall1(int, mysyscall, struct timeval *, thetime)
Main(){
 mysyscall(&tv);
}
```

*现在高版本内核：直接使用syscall()测试使用新添加的系统调用；

4. Linux内核模块和设备驱动程序

参考书：《 Linux Device Drvier 2 》

《 Linux内核编程 》

- 内核模块

LKM Loadable Kernel Modules

- Linux核心是一种monolithic类型的内核，即单一的大核心

另外一种形式是MicroKernel，核心的所有功能部件都被拆成独立部分，这些部分之间通过严格的通讯机制进行联系。

- linux内核是一个整体结构，因此向内核添加任何东西.或者删除某些功能，都十分困难。为了解决这个问题，引入了模块机制，从而可以动态的在内核中添加或者删除模块。模块一旦被插入内核,他就和内核其 他部分一样。

- 模块的实现机制:

Linux为我们提供了两个命令：使用insmod来显式加载核心模块，使用rmmod来卸载模块。同时核心自身也可以请求核心后台进程kerneld来加载与卸载模块。

- 对于每一个内核模块来说，必定包含两个函数:
  - ✓ int init_module() 这个函数在插入内核时启动,在内核中注册一定的功能函数。
  - ✓ int cleanup_module() 当内核模块卸载时，调用它将模块从内核中清除。

参见：“从 2.4 到 2.6：Linux 内核可装载模块机制的改变对设备驱动的影响”

http://www.ibm.com/developerworks/cn/linux/l-module26/

以下是2.6.x核心模块基本框架:

```
#define MODULE
#define __KERNEL__
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/config.h>
#include <linux/version.h>
int init_module(void)
{
printk("<1>Hello,world\n");return 0;
}
void cleanup_module(void)
{
printk("<1>Goodbye cruel world\n");
}
```

kernel 2.6.6 模块定义

kernel 2.6.9 模块定义

```
#define MODULE
#define __KERNEL__
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/config.h>
#include <linux/version.h>
int init_module_func(void)
{
printk("<1>Hello,world\n");return 0;
}
void cleanup_module_func(void)
{
printk("<1>Goodbye cruel world\n");
}
module_init(init_module_func);
module_exit(cleanup_module_func);
MODULE_LICENSE("GPL");
```

通用的Makefile

```
ifneq ($(KERNELRELEASE),)
#kbuild syntax.
mymodule-objs :=test.o        //模块的文件组成
obj-m :=mymodule.o        //生成的模块文件名
else
PWD :=$(shell pwd)
KVER :=$(shell uname -r)
KDIR :=/lib/modules/$(KVER)/build
all:
        $(MAKE) -C $(KDIR) M=$(PWD)
clean:
        rm -f *.cmd *.o *.mod *.ko
endif
```

问题：为什么要用printk？

只能使用内核提供的函数

- 模块为了使用所需核心资源（函数和变量）所以必须能够找到它们。例如模块需要调用核心内存分配函数kmalloc()来分配 内存。模块在加载前并不知道kmalloc()在内存中何处，这样核心必须在加载这些模块前修改模块中对 kmalloc()的引用地址。核心在其核心符号表中维护着一个核心资源链表这样当加载模块时它能够解析出模块 中对核心资源的引用。
- 当试图卸载某个模块时，核心需要知道此模块是否已经没有被使用，同时它需要有种方法来通知此将卸载模块。 模块必须能够在从核心种删除之前释放其分配的所有系统资源，如核心内存或中断。当模块被卸载时，核心将从核心符号表中删除所有与之对应的符号。

- Linux内核中的设备驱动程序是一组常驻内存的具有特权的共享库，是低级硬件处理例程。
- 对用户程序而言，设备驱动程序隐藏了设备的具体细节， 对各种不同设备提供了一致的接口， 一般来说是把设备映射为一个特殊的设备文 件，用户程序可以象对其它文件一样对此设备文件进行操作。
- Linux支持3种设备：字符设备、块设备和网络设备。设备由一个主设备号和一个次设备号标识。主设备号唯一标识了设备类型， 即设备驱动程序类型，它是块设备表或字符设备表中设备表项的索引。次设备号 仅由设备驱动程序解释 .,一般用于识别在若干可能的硬件设备中，I/O请求所涉 及到的那个设备。

- 一个典型的驱动程序,大体上可以分为这么几个部分:
- 注册设备:

  在系统初启,或者模块加载时候,必须将设备登记到相应的设备数组,并返回设备的主设备号;
- 定义功能函数:

  对于每一个驱动函数来说，都有一些和此设备密切相关的功能函数。以最常用的块设备或者字符设备来说,都存在着诸如 open()、read()这一类的操作。当系统调用这些调用时，将自动的使用驱动函数中特定的模块。来实现具体的操作;
- 卸载设备:

  在不用这个设备时,可以将它卸载，主要是从/proc 中取消这个设备的特殊文件。

- 实现
- linux中的大部分驱动程序,是以模块的形式编写的，可以像内核模块一样在需要的时候动态加载，不使用时卸载;
- 注册设备：register_chrdev(…)在init_module()中调用此函数用来注册设备。
- 卸载设备：unregister_chrdev(…)在cleanup_module()中调用此函数用来卸载设备。

int register_chrdev(unsigned int major, const char * name, struct file_operations *fops)

int unregister_chrdev(unsigned int major, const char * name)

用户进程是通过设备文件同硬件打交道：

```
struct file_operations {  //定义在Fs.h
loff_t (*llseek) (struct file *, loff_t, int);//用来修改文件当前的读写位置
ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);//从设备中同步读取数据
ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);//向设备发送数据
int (*flush) (struct file *, fl_owner_t id);
int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long); //执行设备I/O控制命令
int (*open) (struct inode *, struct file *); //打开
int (*release) (struct inode *, struct file *); //关闭
...
 };
```

```
unsigned int test_major = 0;
static int open_test(struct inode *inode,struct file *file )
{ …return 0; }
static void release_tibet(struct inode *inode,struct file *file )
{ … }
struct  file_operations test_fops = {
     .open            = open_test,
     .release         = release_tibet,};

int init_module(void) {  //2.6.6与2.6.9见前面的模块定义
int result;
result = register_chrdev(0, "test", &test_fops);
 if (result < 0) return result;
 if (test_major == 0) test_major = result;  return 0;
}
void cleanup_module(void)
{ unregister_chrdev(test_major, "test");
}
```

- 编译驱动程序并安装

gcc -O2 -DMODULE -D__KERNEL__ -c test.c -o test.o

insmod -f test.o

在2.6.X中，使用前面的Makefile直接make就可以

如果安装成功，在/proc/devices文件中就可以看到设备test, 并可以看到它的主设备号,。

cat /proc/devices | awk "\\$2==\"test\" {print \\$1}"

- 创建设备文件

mknod /dev/test c major minor（c是指创建字符设备 ）

Major主设备号 minor 从设备号（设为0）

- 测试驱动程序

```
Void main(){
testdev = open("/dev/test",O_RDWR);
…..
}
```

- 内存申请和释放

kmalloc()和kfree()。用于在内核模式下申 请和释放内存。

```
void *kmalloc(unsigned int len,int priority);
void kfree(void *__ptr);
```

与用户模式下的malloc()不同，kmalloc()申请空间有大小限制。长度是2的整 次方。可以申请的最大长度也有限制。另外kmalloc()有priority参数，通常使用 时可以为GFP_KERNEL，如果在中断里调用用GFP_ATOMIC参数，因为使用GFP_KERNEL 则调用者可能进入sleep状态，在处理中断时是不允许的。kfree()释放的内存必须是kmalloc()申请的。如果知道内存的大小，也可以用kfree_s()释放。

- 用户空间和内核空间的数据交换

verify_area:

运行在核心态的进程经常需要访问用户地址空间的内容，为了保护内核不受错误信息的攻击，需要验证这些从用户空间传入的地址信息的正确性。

copy_to_user, put_user:

将数据从内核空间移到用户空间

copy_from_user,get_user:

将数据从用户空间移到内核空间

- 申请中断和释放中断的调用

request_irq()、free_irq()

- I/O端口的存取使用：

inb(); inb_p(); outb(); outb_p();

- 打印信息

int printk(const char* fmt, ...);

调试的主要手段

5、proc文件系统

Linux的PROC文件系统是进程文件系统和内核文件系统的组成的复合体,是将内核数据对象化为文件形式进行存取的一种内存文件系统, 是监控内核的一种用户接口. 它拥有一些特殊的文件（纯文本），从中可以获取系统状态信息。

（1）系统信息

与进程无关,随系统配置的不同而不同.

命令procinfo可以显示这些文件的大量信息

（2）进程信息

系统中正在运行的每一个用户级进程的信息。

系统信息

- /proc/cmd/line:内核启动的命令行
- /proc/cpuinfo:CPU信息
- /proc/stat:CPU的使用情况、磁盘、页面、交换、所有的中断、最后一次的启动时间等。
- /proc/meminfo:内存状态的有关信息。

进程信息

- /proc/$pid/stat
- /proc/$pid/status
- /proc/$pid/statm

…….etc

监控系统的功能

- 通过读取proc文件系统，获取系统各种信息，并以比较容易理解的方式显示出来。
- 使用 GTK+ /Qt 下的c语言开发。
- 具体包括：

功能清单

- (1)获取并显示主机名
- (2)获取并显示系统启动的时间
- (3)显示系统到目前为止持续运行的时间
- (4)显示系统的版本号
- (5)显示cpu的型号和主频大小
- (6)同过pid或者进程名查询一个进程，并显示该进程的详细信息，提供杀掉该进程的功能。

功能清单（续）

- (7)显示系统所有进程的一些信息，包括pid，ppid，占用内存大小，优先级等等
- (8)cpu使用率的图形化显示(2分钟内的历史纪录曲线)
- (9)内存和交换分区(swap)使用率的图形化显示(2分钟内的历史纪录曲线)
- (10)在状态栏显示当前时间
- (11)在状态栏显示当前cpu使用率

功能清单（续）

- (12)在状态栏显示当前内存使用情况
- (13)用新进程运行一个其他程序
- (14)关机功能

----参照WINDOWS的任务管理器，实现其中的几个功能。

与proc文件系统中特定文件的关系

- 功能(1)：/proc/sys/kernel/hostname
- 功能(2)：/proc/uptime
- 功能(3)：/proc/uptime
- 功能(4)：/proc/sys/kernel/ostype，/proc/sys/kernel/osrelease
- 功能(5)：/proc/cpuinfo

与proc文件系统中特定文件的关系

- 功能(6)：/proc/(pid)/stat
- 功能(7)：/proc/(pid)/stat,
  /proc/(pid)/statm
- 功能(8)：/proc/stat
- 功能(9)：/proc/meminfo
- 功能(10)：未使用proc文件系统
- 功能(11)：/proc/stat

与proc文件系统中特定文件的关系

- 功能(12)：/proc/meminfo
- 功能(13)：未使用proc文件系统
- 功能(14)：未使用proc文件系统

6、小型文件系统

- 用磁盘中的一个文件（大小事先指定）来模拟一个磁盘；
- 确定文件目录项的结构，空白块的管理（每个块＝连续的N个文件字节）；
- 扩充系统调用命令实现文件的操作(字符界面或图形界面：open、 close、 read、 write、 cp、 rm等,字符界面演示。
- 选择支持：多用户、树形目录（mkdir,cd ...），图形界面。

考核要求

1. 必须独立完成课程设计内容，不分小组，不能有相同的拷贝。
2. 上机检查：学生根据老师提出的要求，演示所完成的系统；并回答老师的问题。
3. 按时到课并签到。
4. 成绩（包括课程设计报告）

完成1、2　60—65

完成1、2、3　65—75

完成1、2、3、4/5　75以上

参考资料

Linux的“man”帮助!

《Linux内核2.4/6版源代码分析大全》

《Linux内核源代码分析》

《Linux编程白皮书》

www.google.com

www.csdn.net

……etc.