操作系统课程设计指导书

陈莉君编写

计算机学院软件工程系

2014.3

一．实验目标

本课程设计以 Linux 操作系统为实验平台。要求学生在熟悉 Linux 系统的基础上，掌握 Linux 命令的基本用法，能够利用 Linux 提供的进程系统调用开发一个小型 Shell 系统，并利用 Linux 提供的模块机制，进行内核级的模块开发，进而能够开发简单的驱动程序。

二．实验题目

题目一：小型 Shell 的设计
题目二：内核模块及简单驱动程序开发
题目三：小型文件系统的设计
（以上题目三选二）

三．实验内容

1.题目一：小型 Shell 的设计

（1）实验目的

通过对一个参考程序的分析和学习，掌握如何编写一个 UNIX 风格 Shell（即命令解释程序）。学习如何接受命令、解释命令、执行命令，特别是用一个新的进程来执行程序，以及父进程如何继续子进程的工作。

（2） 准备知识

Shell 基本介绍

每个 shell 都有自己语言的语法和语义。在标准 Linux shell 中，一个命令行具有以下形式：

命令名　参数 1　参数 2……

shell 的工作是找到与命令名对应的可执行程序（命令），为其准备参数，并使用参数执行命令。

shell 程序需具有以下几种功能和健壮性：

a.支持目录检索功能，即文件不存在，继续打印提示符。
b.支持以“&”结束的输入，进行并发执行(前台与后台)。
c.支持输入输出重定向，“<”，“>”为标志符。
d.支持以“|”进行进程间通信操作（管道功能）。
e.支持一定的错误输入处理，例如：多于空格的出现，输入命令不存在，空输入等等。

考虑 shell 为了完成工作必须采取以下步骤：

a.打印提示符
b.得到命令行
c.解析命令
d.查找文件
e.准备参数
f.执行命令

Shell 工作原理

用户键入的命令可能有误，即便找到要求的可执行文件也可能包含致命的错误，因此要求 shell 程序每接受一个命令后，均创建子进程，让子进程来执行命令文件。如果命令正确，并创建子进程成功，父进程等待子进程完成。如果命令不正确或可执行文件有错或创建子进程失败，则给出相应的错误提示，撤消子进程，回到 shell，等待用户输入下一个命令。这样 shell 进程本身不会受到伤害。

相关系统调用

①int fork(void)

创建一个新进程。调用 fork 的进程称为父进程，新进程是子进程。Fork 系统调用为父子进程返回不同的值：子进程中返回 0，父进程中返回子进程的 PID，如创建不成功返回负数值。

②exec 系列

exec 系统调用用新程序覆盖调用它的进程的地址空间。exec 把一个新的程序装入调用进程的内存空间，来改变调用进程的执行代码。当 exec（）返回时，进程从新程序的第一条指令恢复执行。exec 没有建立一个与调用进程并发执行的新进程，而是用新进程取代了老进程。exec 加后缀，可有多种格式:

```
execl(path,arg0,arg1,…argn,(char *)0);
execv(path,argv);
execlp(file,arg0,arg1,…argn, (char *)0);
execvp(file,argv);
```

其中 l 代表长格式,v 代表利用 argv 传参,e 代表从 envp 传递环境变量,p 代表从 PATH 指定路径搜索文件。

③wait 系列

wait 系统调用可以使进程等待子进程终止。该函数将阻塞调用进程，直到该进程的某一个子进程结束运行（如果子进程收到一个信号而结束运行，该函数也会返回）。子进程的结束状态保存在 status 指向的整数中。如果 status 为空，就不会返回子进程的结束状态。如果在调用 wait 之前子进程结束运行，wait 会立即返回子进程的结束状态，wait 返回值为子进程的进程 ID。若子进程不存在，则返回-1。

```
wait(int *stat_loc);              /*System V,BSD,and POSIX.1 */
wait2(stausp,options,rusagep);       /* BSD */
waitpid(pid,*stat_loc,options);      /* POSIX.1 */
waited(idtype,id,infop,options);     /* SVR2 */
```

以上三个系统调用联用时，形成 shell 用于执行命令的一个代码框架：

```
if(fork()==0){
  execvp(full_pathname,command->argv,0);
}else{
  wait(status);
}
```

④int dup(int fd)

把一个程序的标准输出连接到管道的写入端，把另一个程序的标准输入连接到管道的读出端。返回一个新的文件描述符，该描述符和 fd 指向同一个文件。新的文件描述符具有同样的存取模式，以及同样的读/写偏移量。用这个函数可以进行输入/输出重定向。

输出重定向步骤如下：

```
fid=open(file,O_WRONLY|O_CREAT);
close(1);
dup(fid);
close(fid);
```

标准输入(stdin)、标准输出(stdout)和标准错误(stderr)分别占用 0、1 和 2。要求先关闭标准输出，从而使 1 成为当前系统内可用作文件描述符的最小数（注意，0 被标准输入占用），然后调用 dup，使管道文件与文件描述符 1（即标准输出）连接。

⑤int pipe(int fd[2])

创建管道。该函数可以创建两个文件描述符，fd[0]用于读，fd[1]用于写。这两个文件描述符连接起来就是一个管道。

⑥int open(const char *path,int oflag)

打开文件，path 参数是一个字符串，包含要打开文件的路径，oflag 是标志集合，控制文件的打开方式。可用标志有：

O_RDONLY　只读方式打开
O_WRONLY　只写方式打开
O_RDWR　读/写方式打开
O_CERAT 如果文件已存在，则该选项不执行任何操作。

⑦int close(int fd)

关闭文件。如果关闭成功，返回 0，发生错误返回-1。

⑧int access(const char *path,int amode)

测试路径是否存在。参数 path 中包含了需要检查的文件路径名，amode 是下面几个常量进行或操作的结果：

R_OK 测试文件是否可读；
W_OK 测试文件是否可写；
X_OK 测试文件是否可执行；
F_OK 测试文件是否存在；

分离环境变量中的 PATH 参数可使用如下语句：

```
paths=(char *)getevn(“PATH”);
```

PATH 中路径用“:”分割，可分别取出每个路径，后面再加上文件的相对路径名组成绝对路径，便可使用 access 函数进行检测。

⑨char getenv(char *name)

获得特定环境变量。参数 name 应当是要获取的环境变量的名字。如果该变量存在，就返回该变量的值；否则返回 NULL。

(3) 问题指导

问题 A：自己编写的 shell 命令行解释程序，所执行的结果必须和系统再带的 shell 执行结果保持一致，从而使学生了解系统是怎样进行命令的解析和执行的。同时，还提供认识进程创建、进程等待、进程执行、前/后台执行、管道和 I/O 重定向含义的机会。

基本运行方式如下:

```
identifier [identifier[identifier……]]
```

shell 应该解析命令行参数指针数组 argv[argc]。使用 Linux 的系统调用 fork()、wait()和 execv()等完成。

问题 B：对用户编写的 shell 增加后台运行功能。即用户可以使用”&”作为一个命令结束，以启动下一个命令。

问题 C：修改程序，增加 I/O 重定向功能，可以使用”<”、”>”和”|”符号改变程序/文件的输入和输出。

(4)程序编写指导

a.提示符的打印

在打印提示符之前要使用 get_current_dir_name()得到当前路径。返回一个指向当前目录绝对路径的字符串指针，然后在 stdout 中输出。

b.用户命令和参数的解析

命令和参数被保存在字符串数据 argv[argc]中。在这里要区分命令和参数，其中 argv[0]是命令，程序要识别一般命令并并执行；能识别“>”、“<”和“|”，根据不同的符号转入到不同的程序模块，执行不同的过程。普通的命令在主函数 main()中执行即可，当遇到“>”和“<”是转移到子函数 redirect()执行；当遇到“|”则转移到子函数 pipe()执行。

c.命令文件的获取

子函数 is_fileexit()用来查找命令是否存在。Shell 为每个用户提供了一组环境变量。这些变量定义在用户的.login 文件中。其中路径变量 PATH 是一组绝对路径的列表，表明 shell 如何搜索命令文件。只要我们获取了路径变量，然后依次搜索各个路径，就可以确定用户输入的命令文件的位置了。Leave 指令用来退出自己的 shell 回到 linux 的 shell 下。

d.用户命令的执行

通过调用 fork()创建一个子进程，在子进程中执行命令。在子进程中通过使用 execv()函数来执行命令。

e.重定向的实现

首先使用 open()函数创建一个文件描述符,然后将其复制到文件描述表第二项。这样该进程的所有输出都写到重定向的文件中。使用 dup(fid)函数将标准输出重定向到 fd_out 上（就是重定向文件中）。

f.管道的实现

通过调用函数 pipe(pipeID)，则 pipeID[0]是一个文件描述符，指向管道的读端；对应的，pipeID[1]是一个指向管道写端的指针。创建第一个子进程执行管道符前的命令，并将输出写到管道。然后，创建第二个进程执行管道符后的指令，并从管道读输入流。

(5)参考程序分析指导

① 主要文件

main.c	基本调度 shell 的执行流程
function.c	包括了 shell 功能里的各个功能子模块
head.h	涵盖了要使用的头文件和基本的全局变量、数据结构
shellexe	shell 的可执行文件

d1,d2,testresult 为测试时，临时生成的文件

②主要函数

main（） 负责调度整个 shell 执行的流程

init_shell（） 简单初始化 shell 执行前的 signal

init_command（） 初始化 shell 执行命令前的全局变量初值

get_string（） 从 screen 读取一个字符串

set_background( )检测是否存在&, 设置 background 标志符

delete_space（） 过滤掉输入字符串中的多余的空格,包括前后和连续的多个空格

split（） 将输入字符串，分拆成一个个单词

fill_cmds（）将单词存储到 shellcmd 结构、infile、outfile 文件中去

searchfile（） 分离环境变量 PATH 中设定的路径，调度 scanfile（）

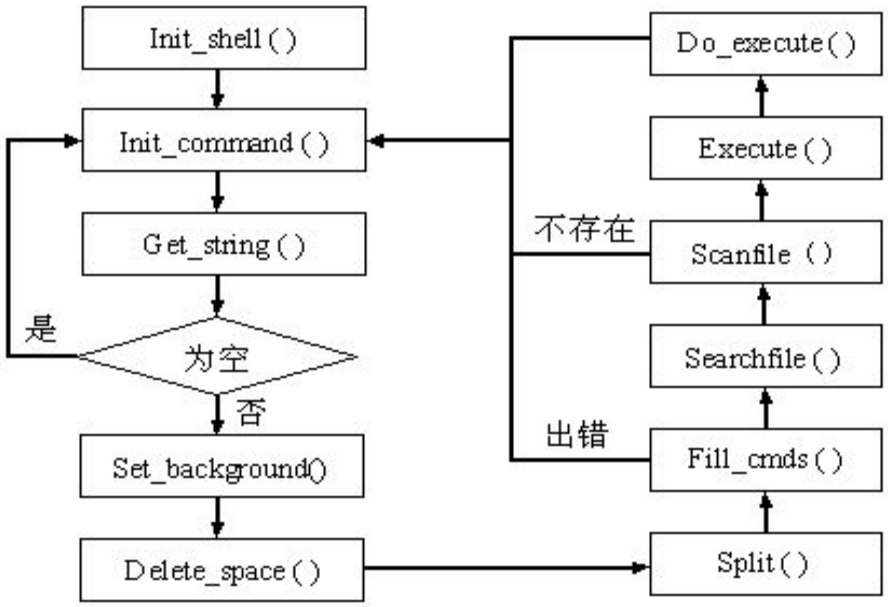
scanfile（） 在当前路径和 PATH 设定的路径中，检索每一个命令是否存在

execute（） 设置 infile,outfile，循环调度 do_execute（）

do_execute（） 执行每一个 shellcmd 中的命令

head.h 包含了要使用的头文件和所有的全局变量和数据结构

③函数调用流程



程序调用流程图

注：参考书中的Linux操作系统内核实习教材详细讲解了原理、要求，并**给出了程序框架**。学生需要仔细阅读这部分内容，就可以很容易地掌握主程序框架的编写方式。实际上需要查询的文件，如何编写程序，在这一节均有提示。以上程序流程图仅作为编程参考。其中含有重复或不尽合理的地方。编程实现时，可以自己另外划分函数与模块功能。

要求学生编写的shell，符合所使用机器安装Linux系统的shell风格。因此，学生要先熟悉Linux的基本命令，命令行方式、参数等，再按要求完成程序功能。

有能力的学生，可以根据shell的基本语法和规则，结合编译知识，考虑试着在一个命令行中，同时实现“>、“<”、“|”、“&”等功能，或同时实现多管道、多重定向等功能。

2 题目二：内核模块及简单驱动程序开发

(1) 实验目的

学习模块机制。这是现代操作系统常用的功能。程序员可用模块动态地增加内核的功能。编写一个模块，将它作为 Linux 内核空间扩展来执行，并报告内核的 xtime 变量值。

(2)准备知识

内核模块是 Linux 内核向外部提供的一个插口，其全称为动态可加载内核模块（Loadable Kernel Module，LKM），我们简称为模块。Linux 内核之所以提供模块机制，是因为它本身是一个单内核（monolithic kernel）。单内核的最大优点是效率高，因为所有的内容都集成在一起，但其缺点是可扩展性和可维护性相对较差，模块机制就是为了弥补这一缺陷。

什么是模块？

模块是具有独立功能的程序，它可以被单独编译，但不能独立运行。它在运行时被链接到内核作为内核的一部分在内核空间运行，这与运行在用户空间的进程是不同的。模块通常由一组函数和数据结构组成，用来实现一种文件系统、一个驱动程序或其他内核上层的功能。

如何编写一个简单的模块？

模块和内核都在内核空间运行，模块编程在一定意义上说就是内核编程。因为内核版本的每次变化，其中的某些函数名也会相应地发生变化，因此模块编程与内核版本密切相关。我们在本书中所涉及的内核编程，基于的内核为 2.6.x，对于其他版本，还需要做一些适当调整。

①程序举例

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

static int __init lkp_init( void )
{
printk("<1>Hello,World! from the kernel space...\n");
return 0;
}

static void __exit lkp_cleanup( void )
{
printk("<1>Goodbye, World! leaving kernel space...\n"); }
module_init(lkp_init);
module_exit(lkp_cleanup);
MODULE_LICENSE("GPL");
```

②说明

a. module.h 头文件中包含了对模块的结构定义以及模块的版本控制，任何模块程序的编写都要包含这个头文件; 头文件 kernel.h 包含了常用的内核函数；而头文件 init.h 包含了宏_init 和_exit，宏_init 告诉编译程序相关的函数和变量仅用于初始化，编译程序将标有_init 的所有代码存储到特殊的内存段中，初始化结束后就释放这段内存。

b.函数 lkp_init ()是模块的初始化函数，它必需包含诸如要编译的代码、初始化数据结构等内容。函数 lkp_cleanup ()是模块的退出和清理函数。

c.我们在这里使用了 printk()函数，该函数是由内核定义的，功能与 C 库中的 printf()类似，它把要打印的信息输出到终端或系统日志。字符串中的<1>是输出的级别，表示立即在终端输出。

d. 函数 module_init()和 cleanup_exit()是模块编程中最基本也是必须的两个函数。module_init()向内核注册模块所提供的新功能，而 cleanup_exit()注销由模块提供的所有功能。

e. 最后一句告诉内核该模块具有 GNU 公共许可证。

③编译模块

假定我们给前面的程序起名为“hellomod.c”，只有超级用户才能加载和卸载模块。对于 2.6 内核的模块，其 Makefile 文件的基本内容如下：

```
obj-m += hellomod.o
all:
    make -C /usr/src/linux -2.6.16 M=$(PWD) modules
clean:
    make -C /usr/src/linux-2.6.16  M=$(PWD) clean
```

上面的 Makefile 中使用了 obj-m := 这个赋值语句，其含义说明要使用目标文件 hellomod.o 建立一个模块，最后生成的模块名是 helloworld.ko，如果你有一个名为 module.ko 的模块依赖于两个文件 file1.o 和 file2.o，那么我们可以使用 module-obj 扩展，如下所示

```
obj-m := module.o
module-objs := file1.o file2.o
```

另外，随发布版的不同，/usr/src/linux-2.6.16 路径中“linux-2.6.16”这个子目录有所不同，请注意区分。关于 Makefile 的具体编写方法，请参考相关书籍。

最后，用 make 命令运行 Makefile

④运行模块代码

当编译好模块，我们就可以将新的模块插入到内核中，这可以用 insmod 命令来实现，如下所示：

```
insmod hellomod.ko
```

然后，可以用 lsmod 命令检查模块是否正确插入到内核中了。

模块的输出由 printk（）来产生。该函数默认打印系统文件/var/log/messages 的内容。快速浏览这些消息可输入如下命令：

```
tail /var/log/messages
```

这一命令打印日志文件的最后 10 行内容，可以看到我们的初始化信息：

```
. . .
. . .
Mar   6 10:25:55   lkp1   kernel: Hello,World! from the kernel space...
```

使用 rmmod 命令，加上我们在 insmod 中看到的模块名，可以从内核中移除该模块（还可

以看到退出时显示的信息)。如下所示：

```
rmmod hellomod
```

同样，输出的内容也在日志文件中，如下所示：

```
. . .
. . .
Mar  6 12:00:05  lkp1  kernel: Hello,World! from the kernel space...
```

⑤应用程序与内核模块的比较

模块编程属于内核编程，因此，除了对内核相关知识有所了解外，还需要了解与模块相关的知识。

为了加深对内核模块的了解，表 1.2 给出应用程序与内核模块程序的比较。

表 1.2 应用程序与内核模块程序的比较

	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	kdbug, kdb,kgdb 等

从表中我们可以看出，内核模块程序不能调用 libc 库中的函数，因为它运行在内核空间，且只有超级用户可以对其运行。另外，模块程序必须通过 module()_init 和 module()_cleanup 函数来告诉内核“我来了”和“我走了”。

⑥proc 文件系统

早期的 Unix 在设备文件目录/dev 下设置了一个特殊文件/dev/mem。通过这个文件可以读/写系统的整个物理内存，这个特殊的文件同样适用于访问磁盘文件的 read()、write()、fseek()等常规的文件操作。采用虚存管理后，Unix 又增加了一个特殊文件/dev/kmem 对应于操作系统的整个虚存空间。这两个特殊文件的作用和体现出的重要性促使人们对其功能加以进一步的扩展，在系统中增设了/proc 目录。经过多年的发展/proc 成为一个特殊的文件系统，因为其中的文件并不是被保存在磁盘上，而是在内存中建立，当对其进行文件读/写操作时才生成相关信息，或者直接映射到系统有关变量或数据结构。文件内容是关于系统中的信息数据，比如，CPU 类型、Linux 的版本信息、内存的使用情况、当前存在的进程、各种设备的运行情况等，它们反映的是计算机的当前状态，所以这些信息会随着系统的运行而呈现不同的数据。这些文件虽然是建立在内存中，但是对它们的访问系统提供了与磁盘文件相同的访问方式，这样可以简化编程。

⑦Linux 中的文件操作

文件的操作步骤通常是先打开或建立文件，然后再对这个文件进行搜索或读/写操作，最后操作完成后关闭文件。在此过程中用到的函数有：

a.打开文件：FILE *fopen(char *filename,char *flags)

返回值：执行成功返回文件指针，执行失败返回 NULL

b.读文本文件：char *fgets(char *buffer,t_size size,FILE *fp)
返回值：指向读得字符缓冲区的指针
c.写文本文件：int fputs(const char *buffer, FILE *fp)
返回值：写入成功返回非负值，否则返回 EOF
d.关闭文件 int fclose(FILE *fp);
返回值：0 表示关闭成功，否则表示失败

(3)问题指导

问题 a：设计并构建一个在/proc 中实现 clock 文件的模块，该文件应该只支持文件的 read()操作。当调用 read()操作时，它返回一个单一的 ASCII 字符串，其中包括用一个空格分割开的两个数字子串，分别表示系统时间变量 xtime 的 tv_sec 和 tv_usec 值。例如：

924280180 184222

系统的时间变量 xtime 设置为:xtime.tv_sec924280180， xtime.tv_usec =184222

问题 b：将编写的模块经过编译，正确加载到内核中。编写一个演示模块的应用程序，使用 gettimeofday()来确定所读取的时钟值和从内核变量中读取的值的明显精度。并学会将该模块从内核中卸载。

问题 c：实验完成后，请解释一下为什么 gettimeofday()会有比计时器中断之间 10 微秒间隔更高的精确度。

(4)程序设计指导

设计主要分为两个部分：

a.设计并构建一个在/proc 中实现 clock 文件的模块 clockmodules。这个模块支持文件的 read()操作。当调用 read()操作时，它返回一个单一的 ASCII 字符串，该字符串包括两个数字串，用空格分开，分别表示系统时间变量 xtime 的 tv_sec 和 tv_usec。将得到的信息放到 proc 文件系统中的一个文件中，以便可以通过程序直接读取信息。

Linux 包含了一种为模块定义 API 的机制，,可以实现模块的相当复杂的功能。模块的实现人员可以为模块的 API 定义任何新函数，所以需要用一些方法通知操作系统这些新函数的存在，这就是模块的注册，通常是在 module_init()中完成。当应用程序想调用这个模块的函数时，就产生一个系统调用，内核根据这个系统调用找到该模块中特定的函数。当模块卸载时，该模块的函数都必须从系统中注销。注销工作通常是在 cleanup_exit()中完成。利用 get_xtime()函数来从内核变量中读取时间。

在 module_init() 函数初始化模块时只需调用一个函数 create_proc_info_entry。这个函数需要传入一个模块名、模块文件权限和一个读取模块的函数指针，系统会自动在系统中创建模块并加载到/proc 文件系统中。而相应的移除模块方法 cleanup_exit()中只需调用 remove_proc_entry 并传入要移除的模块名，系统就会自动将模块从系统中移除。

b.设计一个测试程序（例如名为：timedemo.c）,验证模块。设计一个测试程

序利用 gettimeofday()函数读取时钟值 ,与从内核变量中读取的时间值进行比较。

3 题目三：小型文件系统

(1) 实验内容

为 Linux 系统设计一个简单的二级文件系统,实现文件系统的以下命令：

login	用户登录 （要通过密码验证）
dir	列目录 （要列出文件名，物理地址，保护码和文件长度）
create	创建文件
delete	删除文件
open	打开文件
close	关闭文件
read	读文件
write	写文件

注：需对源文件进行读写权限的保护

(2)程序设计指导

①设计思想

本文件系统采用两级目录，其中第一级对应于用户账号，第二级对应于用户帐号下的文件。另外，为了简便文件系统未考虑文件共享，文件系统安全以及管道文件与设备文件等特殊内容。对这些内容感兴趣的读者，可以在本系统的程序基础上进行扩充。

②主要数据结构

a) i 节点

```
struct inode{
struct inode  *i_forw;
struct inode  *i_back;
char I_flag;
unsigned int i_into;      /*磁盘 i 节点标号*/
unsigned int i_count;     /*引用计数*/
unsigned short di_number; /*关联文件书，当为 0 时，则删除该文件*/
unsigned short di_mode;   /*存取权限*/
unsigned short di_uid;    /*磁盘 i 节点用户*/
unsigned short di_gid;    /*磁盘 i 节点组*/
Unsigned int di_addr[NADDR];}          /*物理块号*/
```

b) 磁盘 i 结点

```
Struct dinode
    {
     unsigned short di_number;        /*关联文件数*/
     unsigned short di_mode;          /*存取权限*/
     unsigned short di_uid;
     unsigned short di_gid;
```

```
     unsigned long di_size;                /*文件大小*/
     unsigned int di_addr[NADDR];          /*物理块号*/
```

c)目录项结构

```
Struct direct
    {
     char d_name[DIRSIZ];              /*目录名*/
     unsigned int d_ino;               /*目录号*/
    }
```

d)超级块

```
Struct filsys
    {
     unsigned short s_isize;            /*i 节点块块数*/
     unsigned long s_fsize;             /*数据块块数*/
     unsigned int s_nfree;              /*空闲块块数*/
     unsigned short s_pfree;            /*空闲块指针*/
     unsigned int s_free[NICFREE];      /*空闲块堆栈*/
     unsigned int s_ninode;             /*空闲 i 节点数*/
     unsigned short s_pinode;           /*空闲 i 节点指针*/
     unsigned int s_inode[NICINOD];     /*空闲 i 节点数组*/
     unsigned int s_rinode;             /*铭记 i 节点*/
     char s_fmod;                      /*超级块修改标志*/
    };
```

e)用户密码

```
Struct pwd
    {
     unsigned short P_uid;
    unsigned short P_gid;
    char passward[PWOSIZ];
    }
```

f) 目录

```
Struct dir
  {
   strut direct direct[DIRNUM];
   int size;
  }
```

g).查找 i 内存节点的 hash 表

```
Struct hinode
  {
   strut inode *iforw;
  }
```

h).系统打开表

```
  Struct file
  {
    char f_flag; /*文件操作标志*/
```

```
    unsigned int f_count; /*引用计数*/
    struct inode *f_inode; /*指向内存节点*/
    unsigned long f_off; /*读/写指针*/
  }
  i)用户打开表
Struct user
  {
    unsigned short u_default_mode;
    unsigned short u_uid; /*用户标志*/
    unsigned short u_gid; /*用户组标志*/
    unsigned short u_ofile[NOFILE]; /*用户打开表*/
  }
```

③主要函数

a.节点内容获取函数 iget()（详细描述略）。
b.点内容释放函数 iput()（详细描述略）。
c.目录创建函数 mkdir()（详细描述略）。
d.目录搜索函数 namei()（详细描述略）。
e.磁盘块分配函数 balloc()（详细描述略）。
f.磁盘块释放函数 bfree()（详细描述略）。
g.分配 i 节点区函数 ialloc()（详细描述略）。
h.解释结点区函数 ifree（ ）（详细描述略）。
i.搜索当前目录下文件的函数 iname（ ）（详细描述略）。
j.访问控制函数 access（ ）（详细描述略）。
k.显示目录和文件函数_dir（ ）（详细描述略）。
l.改变当前目录用函数 chdir（ ）（详细描述略）。
m.打开文件函数 open（ ）（详细描述略）。
n.创建文件函数 create（ ）（详细描述略）。
o.读文件用函数 read（ ）（详细描述略）。
p.读文件用函数 write（ ）（详细描述略）。
q.用户登陆函数 login（ ）（详细描述略）。
r.用户退出函数 logout（ ）（详细描述略）。
s.文件系统格式化函数 format（ ）（详细描述略）。
t.进入文件系统函数 install（ ）（详细描述略）。
u.关闭文件函数 close（ ）（详细描述略）。
v.退出文件系统函数 halt（ ）（详细描述略）。
w.文件删除函数 delecte（ ）（详细描述略）。

④主程序说明

```
begin
    Step1       对磁盘进行格式化
    Step2       调用 install（），进入文件系统
    Step2       调用_dir（），显示当前目录
    Step4       调用 login（），用户注册
    Step5       调用 mkdir（）和 chdir（）创建目录
    Step6       调用 create（），创建文件 0
```

Step7　　分配缓冲区
Step8　　写文件 0
Step9　　关闭文件 0 和释放缓冲
Step10　　调用 mkdir（）和 chdir（）创建子目录
Step11　　调用 create（），创建文件 1
Step12　　分配缓冲区
Step12　　写文件 1
Step14　　关闭文件 1 和释放缓冲
Step15　　调用 chdir 将当前目录移到上一级
Step16　　调用 create（），创建文件 2
Step17　　分配缓冲区
Step18　　调用 write（），写文件 2
Step19　　关闭文件 1 和释放缓冲
Step20　　调用 delecte（），删除文件 0
Step21　　调用 create（），创建文件 1
Step22　　为文件 2 分配缓冲区
Step22　　调用 write（），写文件 2
Step24　　关闭文件 2 并释放缓冲区
Step25　　调用 open()，打开文件 2
Step26　　为文件 2 分配缓冲
Step27　　写文件 2 后关闭文件 2
Step28　　释放缓冲
Step29　　用户退出（logout）
Step20　　关闭（halt）
End

由上述的描述过程可知，该文件系统实际是为用户提供一个解释执行相关命令的环境。主程序中的大部分语句都被用来执行相应的命令。

四．考核方式

成绩考核考虑以下四个方面的内容：

（1）　题目一、三选择一个，二为必做

（2）　对题目扩充的程度及难度

（3）　对 Linux 系统熟悉的程度

（4）　设计结果

（5）　设计报告

成绩计分按优、良、中、及格、不及格 5 级评定。

五．参考书

（1）Linux 操作系统内核实习 冯锐等译 机械工业出版社　2001 年 6 月
（2）Linux 内核设计与实现 陈莉君译 机械工业出版社 2005 年 11 月
（3）深入理解 Linux 内核 陈莉君等译 中国电力出版社 2004 年 6 月

（4）深入分析 linux 内核及其应用 赵亚著 机械工业出版社 2011 年 6 月
（5）linux 内核 api 完全参考手册 邱铁;周玉;邓莹莹著 机械工业出版社 2011 年 01 月
（6）)Linux 内核编程［美］daniel p. bovet, marco cesati(著) | 陈莉君 冯锐 牛欣源(译) 中国电力出版社 2011 年6月

（**适用专业**：软件工程 **版本**：V1.0)

附录：编写 Makefile 文件

```
 /*********************************************
makefile
*********************************************/
```

```
filsys:main.o iallfre.o ballfre.o name.o access.o log.o close.o creat.o delete.o
dir.o
      open.o rdwt.o format.o install.o halt.o cc-o filsys main.o iallfre.o ballfre.o
      name.o access.o log.o close.o creat.o delete.o dir.o open.o format.o install.o
halt.o
main.o:main.c filesys.h
       cc-c main.c
igetput.o: igetput.c filesys.h
         cc-c igetput.c
iallfre.o: iallfre.c filesys.h
       cc-c iallfre.c
ballfre.o: ballfre.c filesys.h
       cc-c ballfre.c
name.o:name.c filesys.h
       cc-c name.c
access.o:access.c filesys.h
        cc-c access.c
log.o:log.c filesys.h
     cc-c log.c
close.o:close.c filesys.h
       cc-c close.c
creat.c:creat.c filesys.h
       cc-c creat.c

delete.o:delete.c filesys.h
        cc-c delete.c
dir.o:dir.c filesys.h
   cc-c dir.c
open.o:open.c filesys.h
     cc-c open.c
rdwt.o:rdwt.c filesys.h
       cc-c rdwt.c
format.o:format.c filesys.h
         cc-c format.c
install.o: install.c filesys.h
            cc-c install.c
    halt.o:halt.c
    cc-c halt.c
```