

**操作系统原理课程设计报告**

|  |  |
| --- | --- |
| 姓 名： | 徐立威 |
| 学 院： | 华中科技大学 |
| 专 业： | 信息安全 |
| 班 级： | 1703 |
| 学 号： | U201714513 |
| 指导教师： | 羌卫中 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2020年 6 月 18

目录

[1. 实验一 Linux下C编程的相关知识 1](#_Toc9809)

[1.1. 实验目的 1](#_Toc3091)

[1.2. 实验内容 1](#_Toc26860)

[1.3. 实验设计 1](#_Toc9583)

[1.3.1. 开发环境 1](#_Toc15981)

[1.3.2. 实验设计 2](#_Toc28883)

[1.4. 实验调试 5](#_Toc26290)

[1.4.1. 实验步骤 5](#_Toc3664)

[1.4.2. 实验调试及心得 8](#_Toc10867)

[附录 实验代码 11](#_Toc20741)

[2. 实验二 系统调用相关知识 18](#_Toc4237)

[2.1. 实验目的 18](#_Toc2987)

[2.2. 实验内容 18](#_Toc13023)

[2.3. 实验设计 18](#_Toc25223)

[2.3.1. 开发环境 18](#_Toc22261)

[2.3.2. 实验设计 18](#_Toc21993)

[2.4. 实验调试 19](#_Toc20746)

[2.4.1. 实验步骤 19](#_Toc4508)

[2.4.2. 实验调试及心得 24](#_Toc11314)

[附录 实验代码 28](#_Toc12542)

[3. 实验三 设备驱动相关知识 29](#_Toc10401)

[3.1. 实验目的 29](#_Toc25743)

[3.2. 实验内容 29](#_Toc26993)

[3.3. 实验设计 29](#_Toc3382)

[3.3.1. 开发环境 29](#_Toc13654)

[3.3.2. 实验设计 29](#_Toc17050)

[3.4. 实验调试 33](#_Toc2584)

[3.4.1. 实验步骤 33](#_Toc9522)

[3.4.2. 实验调试及心得 36](#_Toc213)

[附录 实验代码 37](#_Toc8709)

[4. 实验四 /proc文件相关知识 42](#_Toc12631)

[4.1. 实验目的 42](#_Toc15661)

[4.2. 实验内容 42](#_Toc29946)

[4.3. 实验设计 42](#_Toc24978)

[4.3.1. 开发环境 42](#_Toc11775)

[4.3.2. 实验设计 43](#_Toc26501)

[4.4. 实验调试 45](#_Toc7745)

[4.4.1. 实验调试及心得 45](#_Toc20761)

[附录 实验代码 47](#_Toc28824)

[5. 实验五 文件系统 72](#_Toc29302)

[5.1. 实验目的 72](#_Toc29310)

[5.2. 实验内容 72](#_Toc11632)

[5.3. 实验设计 72](#_Toc10158)

[5.3.1. 开发环境 73](#_Toc25384)

[5.3.2. 实验设计 73](#_Toc28622)

[5.4. 实验调试 73](#_Toc5005)

[5.4.1. 实验调试及心得 73](#_Toc30315)

[附录 实验代码 80](#_Toc8684)

# 实验一 Linux下C编程的相关知识

## 实验目的

* 掌握添加系统调用的方法

## **实验内容**

* 编写一个C程序，用read、write等系统调用实现文件拷贝功能。命令形式：

copy <源文件名> <目标文件名>

* 编写一个C程序，使用图形编程库 (QT/GTK)分窗口显示三个并发进程的运行(一个窗口实时显示当前系统时间，一个窗口循环显示0到9，一个窗口做1到1000的累加求和，刷新周期均为1秒)。

## **实验设计**

## 开发环境

* 主机系统版本：Window 10 64位
* 主机内存：8G
* Intel(R) Core(TM) i5-4300M CPU @2.60GHz 2.60 GHz
* 虚拟机：vmware workstation
* 虚拟机操作系统 Ubuntu16.04 64位
* 虚拟机内存：4G
* 虚拟机硬盘大小：20G
* 图形界面：Gtk

## 实验设计

#### 文件拷贝

首先先设置好输入格式，我们设置好命令格式为：执行文件 被拷贝文件 拷贝文件。如果输入的参数个数不等于3，就报错。这样就成功的把文件的地址存入argv[1]和argv[2]，以便后续步骤的实现。

学会open、close、read、write四个函数的使用，对于open的打开，指导书上进行了详细的介绍。open函数有两个形式，如下所示：

int open(const char \*pathname,int flags);

int open(const char \*pathname,int flags,mode\_t mode);

其中地址位置是固定的，两种方式都需要，其次就是打开文件权利的需要，就是flags，flags的选择如下所示：

O\_RDONLY:以只读的方式打开文件. //是字母O，

O\_WRONLY:以只写的方式打开文件.

O\_RDWR:以读写的方式打开文件.

O\_APPEND:以追加的方式打开文件.

O\_CREAT:创建一个文件. //创建一个不存在的文件。

O\_EXEC:如果使用了O\_CREAT而且文件已经存在,就会发生一个错误.

O\_NOBLOCK:以非阻塞的方式打开一个文件.

O\_TRUNC:如果文件已经存在,则删除文件的内容.

前面三个标志只能使用任意的一个。

显然源文件的打开只需要只读权利，因此只要用open(argv[1],O\_RDONLY)打开即可，而写入文件要考虑不存在的情况，并且一定需要写入操作，因此，我们可以使用open(argv[2],O\_WRONLY|O\_CREAT,S\_IRUSR|S\_IWUSR)来打开。

如果打开文件失败，则退出，并显示哪个文件打不开，以便之后的问题发现。

文件打开之后，就是要实现文件的拷贝与复制，指导书中有一个经典的拷贝算法，如下所示：

while(bytes\_read=read(from\_fd,buffer,BUFFER\_SIZE))

{

/\* 一个致命的错误发生了 \*/

if((bytes\_read==-1)&&(errno!=EINTR)) break;

else if(bytes\_read>0)

{

ptr=buffer;

while(bytes\_write=write(to\_fd,ptr,bytes\_read))

{

/\* 一个致命错误发生了 \*/

if((bytes\_write==-1)&&(errno!=EINTR))break;

/\* 写完了所有读的字节 \*/

else if(bytes\_write==bytes\_read) break;

/\* 只写了一部分,继续写 \*/

else if(bytes\_write>0)

{

ptr+=bytes\_write;

bytes\_read-=bytes\_write;

}

}

/\* 写的时候发生的致命错误 \*/

if(bytes\_write==-1)break;

}

}

至此，读写操作完成。即可关闭文件，结束进程。

之后就是简单的编译过程，gcc -o main main.c。当然，到了后面写gtk的编译指令时可以写个makefile，来避免指令的记忆，但是这里还是相对的简单，所以没必要使用。

#### 并发进程

首先要做的是装好gtk，这个部分会在实验步骤中详细的介绍，这里就略过。在装完gtk后，就开始代码的编写。

我在gtk的掌握方面很少，属于那种初学的情况。但从我课设的学习时间，我大概总结下设计的过程。Gtk的编写大多是两种方法，一是纯代码编写，二是借助glade的图形编辑工具写把要的框架打好，之后再把代码实现的功能与之链接上。

第一次实验并不是十分的复杂，所以，我采用了第一种，纯代码编写的方法。为了简单起见，参考了一下一个完整的gtk代码，有了一个总的轮廓。

main.c：

int pid =fork ();

if (pid < 0)

printf ("error!\n");

首先我们创建一个子进程，如果创建失败（即资源分配失败）就报错。

else if (pid == 0)

{

int pid = fork ();

if (pid < 0)

printf ("error!\n");

如果创建的子进程成功，则在该子进程下再创建一个子进程。

else if (pid == 0)

show1 (argc,argv,"process1");

else

show2 (argc,argv,"process2"); }

else

show3 (argc,argv,"process3");

显示3个窗口，在main函数写完后，就是对3个窗口的功能编写，分别实现一个显示时间，显示0-9的轮流变化，显示1-1000的累加求和，如下所示：

1. 功能一：显示时间
2. 功能二：显示0-9的轮流变化
3. 功能三：显示1-1000的累加求和

## 实验调试

## 实验步骤

#### 任务一：文件拷贝

实验一的任务一是对LINUX操作系统的函数调用的熟悉，学会源程序的编译，下面就进行简单的C语言的源程序编译，实现文件拷贝功能，这里我们要使用gcc编译器。具体步骤如下：

1. 编写代码

实验指导书已经为我们准备了实验代码，主要用到如下几个函数：

open：打开文件

close：关闭文件

read：读操作

write：写操作

当我们需要打开一个文件进行读写操作的时候,我们可以使用系统调用函数open.使用完成以后我们调用另外一个close函数进行关闭操作.

#include <fcntl.h> //注意：这个字母是小写的L，而不是数字1。

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

int open(const char \*pathname,int flags);

int open(const char \*pathname,int flags,mode\_t mode);

int close(int fd);

（2）编译代码

利用指令gcc -o task1 task1.c即可对源代码进行编译，如图1.1。

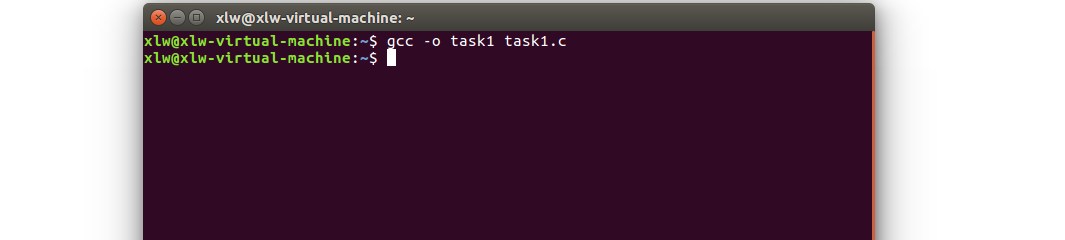


图1.1

#### **LINUX换源**

任务二要用到gtk，需要安装很多的软件，为了提高速度，在做并发进程之前，建议换下源。否则速度会十分的慢，而且不方便下载。

1. 备份

为防止意外，预先备份好原来的源，输入命令：

sudo cp /etc/apt/sources.list /etc/apt/sources\_init.list

1. 更换源

使用gedit打开文档，将下边的阿里源复制进去，然后点击保存关闭。打开文档的命令：

sudo gedit /etc/apt/sources.list

阿里源：

deb http://mirrors.aliyun.com/ubuntu/ trusty main restricted universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ trusty-security main restricted universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ trusty-updates main restricted universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ trusty-proposed main restricted universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ trusty-backports main restricted universe multiverse

deb-src http://mirrors.aliyun.com/ubuntu/ trusty main restricted universe multiverse

deb-src http://mirrors.aliyun.com/ubuntu/ trusty-security main restricted universe multiverse

deb-src http://mirrors.aliyun.com/ubuntu/ trusty-updates main restricted universe multiverse

deb-src http://mirrors.aliyun.com/ubuntu/ trusty-proposed main restricted universe multiverse

deb-src http://mirrors.aliyun.com/ubuntu/ trusty-backports main restricted universe multiverse

1. 更新

主要是两条命令，更新源和更新软件：

更新源：sudo apt-get update

更新软件：sudo apt-get upgrade

#### **gtk的安装配置**

个人看了很多的博客，但是发现其中有许多的问题。在实验过程中，sudo apt-get install gnome-core-devel指令无法运行找不到该库。这可能是指令发生了变化，解决方法是去掉中间的-core，直接运行sudo apt-get install gnome-devel。

下面是指令的具体操作。

　1、安装gcc/g++/gdb/make 等基本编程工具

$sudo apt-get install build-essential

　2、安装 libgtk2.0-dev libglib2.0-dev 等开发相关的库文件

$sudo apt-get install gnome-core-devel （纠正成sudo apt-get install gnome-devel）

　3、用于在编译GTK程序时自动找出头文件及库文件位置

$sudo apt-get install pkg-config

　4、安装 devhelp GTK文档查看程序

$sudo apt-get install devhelp

　5、安装 gtk/glib 的API参考手册及其它帮助文档

$sudo apt-get install libglib2.0-doc libgtk2.0-doc

　6、安装基于GTK的界面GTK是开发Gnome窗口的c/c++语言图形库

$sudo apt-get install glade libglade2-dev

或者

$sudo apt-get install glade-gnome glade-common glade-doc

　7、安装gtk2.0 或者 将gtk+2.0所需的所有文件统通下载安装完毕

$sudo apt-get install libgtk2.0-dev

或者

$sudo apt-get install libgtk2.0\*

　8、查看是否安装了gtk

$pkg-config --list-all grep gtk

#### **任务二：并发进程**

1. 代码编写

代码已经复制到附录当中，参考了很多博客上知识，其中主要是使用进度条控件来实现累加求和计时功能。

1. Makefile文件

由于gtk指令过长，为了方便，我找了个makefile的模板。模板如下所示：

cc＝gcc

all：

（你所要输入的指令）

clean:

rm（生成的程序）

这么做主要是图方便，下次只要输入make即可自动编译。

## 实验调试及心得

#### 文件拷贝

首先我们打开拷贝文件和被拷贝文件，我们命名为new和old，如图1.2。我们可以看到，old文件有内容，new文件是空文件。



图1.2

接下来，执行代码，格式为是./task1 被拷贝文件 拷贝文件。在图1.2中创建的两个文件old，new，如图1.3

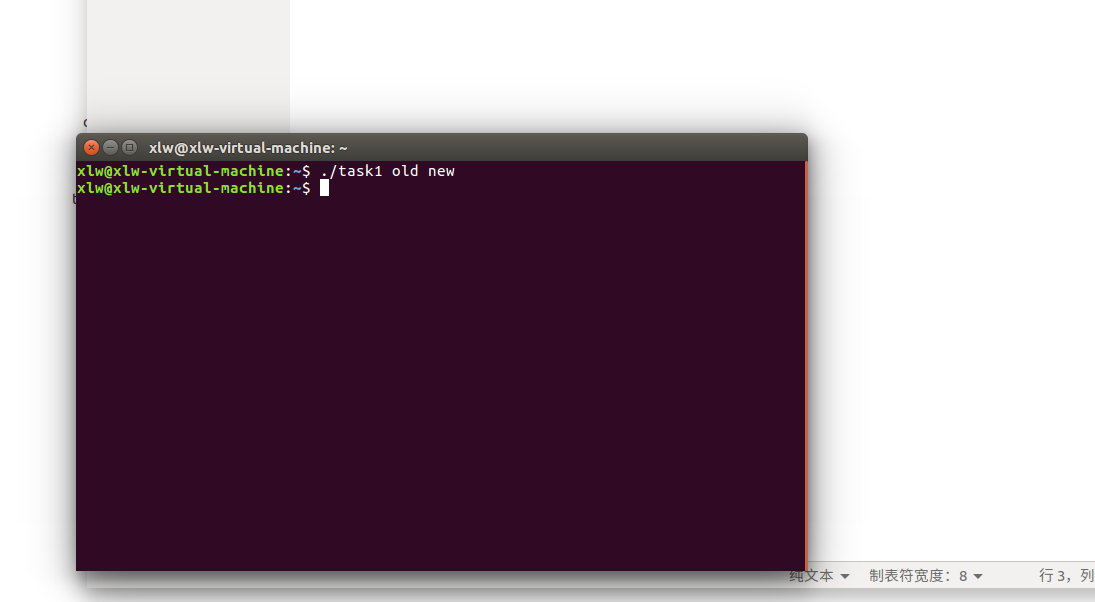


图1.3

执行完拷贝操作后，我们再对new文件进行刷新，发现内容已被拷贝进来。如图1.4。说明拷贝成功。



图1.4

#### 并发进程

展现出3个窗口，分别是0-9的循环，1-1000的累加和当前时间的显示，我就截了其中一张图片，如图1.5。分别对应着process1是当前时间的显示，process2是0-9循环，process3是1-1000的累加求和，三者之间还呈现父子进程的关系，由于gtk知识不是很丰富，所以把累加求和用进度条来实现，主要是觉得将process2的进度条改造成累加求和方便。

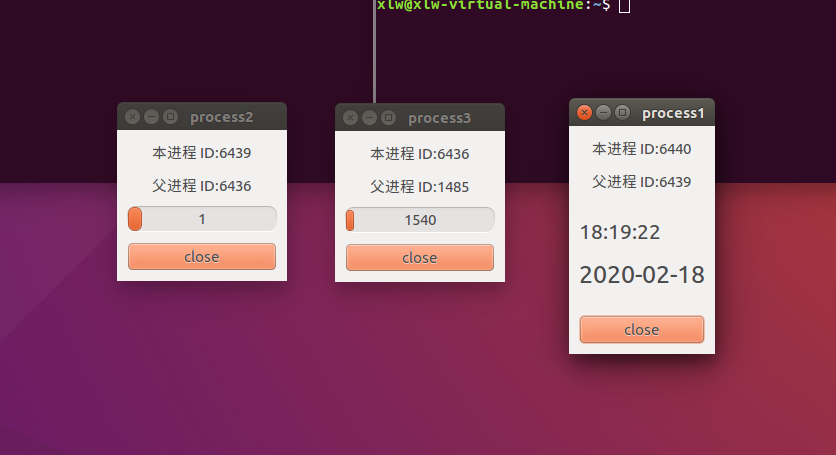


图1.5

#### 实验心得

做完所有的实验之后，再来回味实验一，突然发现这其实只是开始，两个小实验分别给予的是我对LINUX系统操作的重拾以及gtk学习的初步。在此过程中，我通过学习网上的CSDN论坛的经验以及在老师、助教和同学的指导下，解决了LINUX系统换源、gtk安装、gtk实现的图形化界面设计、makefile文件的编写等多个问题。所以，我也想写下自己做实验的经验之谈，

1. 下载速度十分重要

在没有换源之前，那下载速度是真的感人，随便更新个软件、设置之类的，基本上进度条丝毫不变，这样子十分的浪费时间。

1. Gtk 0基础很难受

作为gtk的新手使用者，我开始是模仿别人的代码，在别人的基础上来感受gtk的设计过程。我觉得任何的学习，都是从模仿开始的，我不相信大多数人都能一蹴而就，能在看完gtk手册就能完成毫无误差的代码，这不太可能，也不太现实。所以，在这阶段，好好地模仿别人，学习其中的编写特点，对应每个函数和他的实际利用，从中来悟出自己的经验。

1. 虚拟机环境需要配置好

在LINUX系统的配置上，我感觉个人配置的还算舒服，输入法、中文的配置，vmtool的安装（在换成自己的内核后貌似vmtool会出错，实现不了虚拟机和主机的文件直接传递）等使得自己在使用时还是相当的舒服。

## 附录 实验代码

Task 1拷贝文件源代码：

#include <unistd.h>

#include <fcntl.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <errno.h>

#include <string.h>

#define BUFFER\_SIZE 1024 //缓冲区大小

int main(int argc, char \*\*argv)

{

int from\_fd, to\_fd;

int bytes\_read, bytes\_write;

char buffer[BUFFER\_SIZE]; //设定一个缓冲区

char \*ptr;

if (argc != 3) //三个参数

{

fprintf(stderr, "Usage:%s fromfile tofile\n\a", argv[0]);

return(-1);

}

/\* 打开源文件 \*/

if ((from\_fd = open(argv[1], O\_RDONLY)) == -1)

{

fprintf(stderr, "Open %s Error:%s\n", argv[1], strerror(errno));

return(-1);

}

/\* 创建目的文件 \*/

if ((to\_fd = open(argv[2], O\_WRONLY | O\_CREAT, S\_IRUSR | S\_IWUSR)) == -1)

{

fprintf(stderr, "Open %s Error:%s\n", argv[2], strerror(errno));

return(-1);

}

while (bytes\_read = read(from\_fd, buffer, BUFFER\_SIZE))

{

/\* 出错\*/

if ((bytes\_read == -1) && (errno != EINTR)) break;

else if (bytes\_read>0)

{

ptr = buffer;

while (bytes\_write = write(to\_fd, ptr, bytes\_read))

{

/\* 出错\*/

if ((bytes\_write == -1) && (errno != EINTR))break;

/\* 写完了所有读的字节 \*/

else if (bytes\_write == bytes\_read) break;

/\* 只写了一部分 ,继续写 \*/

else if (bytes\_write>0)

{

ptr += bytes\_write;

bytes\_read -= bytes\_write;

}

}

/\* 写的时候出错 \*/

if (bytes\_write == -1)break;

}

}

close(from\_fd);

close(to\_fd);

return(1);

}

Task 2源代码：

#include <gtk/gtk.h>

#include <sys/types.h>

#include <unistd.h>

gint progress\_timeout( gpointer pbar )

{

gdouble new\_val;

char s[10];

static int i=0;

new\_val = gtk\_progress\_bar\_get\_fraction (GTK\_PROGRESS\_BAR (pbar)) +

0.1;

i++;

if (new\_val > 0.9)

{

i=0;

new\_val = 0.0;

}

sprintf (s, "%d", i);

gtk\_progress\_bar\_set\_fraction (GTK\_PROGRESS\_BAR (pbar), new\_val);

gtk\_progress\_bar\_set\_text (GTK\_PROGRESS\_BAR (pbar),s);

return TRUE;

}

gint progress\_timeout1( gpointer pbar )

{

static int i=0;

static int sum=0;

gdouble new\_val;

char s[10];

new\_val = gtk\_progress\_bar\_get\_fraction (GTK\_PROGRESS\_BAR (pbar)) +

0.001;

sum+=i++;

if (new\_val > 1.0)

return TRUE;

sprintf (s, "%d", sum);

gtk\_progress\_bar\_set\_fraction (GTK\_PROGRESS\_BAR (pbar), new\_val);

gtk\_progress\_bar\_set\_text (GTK\_PROGRESS\_BAR (pbar),s);

return TRUE;

}

gboolean settime(gpointer data)

{

time\_t times;

struct tm \*p\_time;

time(&times);

p\_time = localtime(&times);

gchar \*text\_data = g\_strdup\_printf(\

"<span size='xx-large'>%04d-%02d-%02d</span>",\

(1900+p\_time->tm\_year),(1+p\_time->tm\_mon),(p\_time->tm\_mday));

gchar \*text\_time = g\_strdup\_printf(\

"<span size='x-large'>%02d:%02d:%02d</span>",\

(p\_time->tm\_hour), (p\_time->tm\_min), (p\_time->tm\_sec));

gchar \*text\_markup = g\_strdup\_printf("\n%s\n\n%s\n", text\_time, text\_data);

gtk\_label\_set\_markup(GTK\_LABEL(data), text\_markup);

return TRUE;

}

void destroy\_progress( GtkWidget \*widget)

{

gtk\_main\_quit ();

}

void show1(int argc,char \*argv[],char \*title )

{

GtkWidget \*window;

GtkWidget \*label;

GtkWidget \*vbox;

char id\_char[50];

GtkWidget \*button;

gtk\_init (&argc, &argv);

window = gtk\_window\_new (GTK\_WINDOW\_TOPLEVEL);

gtk\_window\_set\_resizable (GTK\_WINDOW (window), TRUE);

gtk\_window\_set\_position(GTK\_WINDOW(window),

GTK\_WIN\_POS\_CENTER);

gtk\_window\_set\_title (GTK\_WINDOW(window), title);

vbox = gtk\_vbox\_new (FALSE, 10);

gtk\_container\_set\_border\_width (GTK\_CONTAINER (vbox), 10);

gtk\_container\_add (GTK\_CONTAINER (window), vbox);

gtk\_widget\_show (vbox);

sprintf (id\_char, "本进程 ID:%d", getpid ());

label = gtk\_label\_new (id\_char);

gtk\_box\_pack\_start (GTK\_BOX (vbox), label, FALSE, FALSE, 0);

gtk\_widget\_show (label);

sprintf (id\_char, "父进程 ID:%d", getppid ());

label = gtk\_label\_new (id\_char);

gtk\_box\_pack\_start (GTK\_BOX (vbox), label, FALSE, FALSE, 0);

gtk\_widget\_show (label);

label = gtk\_label\_new (NULL);

g\_signal\_connect (G\_OBJECT(window), \

"delete\_event", G\_CALLBACK(gtk\_main\_quit), NULL);

gtk\_container\_add (GTK\_CONTAINER(window), label);

gint s = g\_timeout\_add (1000, settime, (void \*)label);

gtk\_box\_pack\_start (GTK\_BOX (vbox), label, FALSE, FALSE, 0);

gtk\_widget\_show (label);

button = gtk\_button\_new\_with\_label ("close");

g\_signal\_connect\_swapped (G\_OBJECT (button), "clicked", G\_CALLBACK

(gtk\_widget\_destroy), window);

gtk\_box\_pack\_start (GTK\_BOX (vbox), button, FALSE, FALSE, 0);

GTK\_WIDGET\_SET\_FLAGS (button, GTK\_CAN\_DEFAULT);

gtk\_widget\_grab\_default (button);

gtk\_widget\_show (button);

gtk\_widget\_show\_all (window);

gtk\_main();

}

void show3(int argc,char \*argv[],char \*title )

{

GtkWidget \*window;

GtkWidget \*vbox;

GtkWidget \*pbar;

GtkWidget \*pbar2;

GtkWidget \*button;

GtkWidget \*label;

int timer;

char id\_char[50];

gtk\_init (&argc, &argv);

window = gtk\_window\_new (GTK\_WINDOW\_TOPLEVEL);

gtk\_window\_set\_resizable (GTK\_WINDOW (window), TRUE);

gtk\_window\_set\_position(GTK\_WINDOW(window),

GTK\_WIN\_POS\_CENTER);

g\_signal\_connect (G\_OBJECT (window), "destroy", G\_CALLBACK

(destroy\_progress), NULL);

gtk\_window\_set\_title (GTK\_WINDOW (window), title);

gtk\_container\_set\_border\_width (GTK\_CONTAINER (window), 0);

vbox = gtk\_vbox\_new (FALSE, 10);

gtk\_container\_set\_border\_width (GTK\_CONTAINER (vbox), 10);

gtk\_container\_add (GTK\_CONTAINER (window), vbox);

gtk\_widget\_show (vbox);

sprintf (id\_char, "本进程 ID:%d", getpid ());

label = gtk\_label\_new (id\_char);

gtk\_box\_pack\_start (GTK\_BOX (vbox), label, FALSE, FALSE, 0);

gtk\_widget\_show (label);

sprintf (id\_char, "父进程 ID:%d", getppid ());

label = gtk\_label\_new (id\_char);

gtk\_box\_pack\_start (GTK\_BOX (vbox), label, FALSE, FALSE, 0);

gtk\_widget\_show (label);

pbar = gtk\_progress\_bar\_new ();

gtk\_box\_pack\_start (GTK\_BOX (vbox), pbar, FALSE, FALSE, 0);

gtk\_widget\_show (pbar);

timer = gtk\_timeout\_add (1000, progress\_timeout1, pbar);

button = gtk\_button\_new\_with\_label ("close");

g\_signal\_connect\_swapped (G\_OBJECT (button), "clicked", G\_CALLBACK

(gtk\_widget\_destroy), window);

gtk\_box\_pack\_start (GTK\_BOX (vbox), button, FALSE, FALSE, 0);

GTK\_WIDGET\_SET\_FLAGS (button, GTK\_CAN\_DEFAULT);

gtk\_widget\_grab\_default (button);

gtk\_widget\_show (button);

gtk\_widget\_show (window);

gtk\_main ();

}

void show2(int argc,char \*argv[],char \*title )

{

GtkWidget \*window;

GtkWidget \*vbox;

GtkWidget \*pbar;

GtkWidget \*pbar2;

GtkWidget \*button;

GtkWidget \*label;

int timer;

char id\_char[50];

gtk\_init (&argc, &argv);

window = gtk\_window\_new (GTK\_WINDOW\_TOPLEVEL);

gtk\_window\_set\_resizable (GTK\_WINDOW (window), TRUE);

gtk\_window\_set\_position(GTK\_WINDOW(window),

GTK\_WIN\_POS\_CENTER);

g\_signal\_connect (G\_OBJECT (window), "destroy", G\_CALLBACK

(destroy\_progress), NULL);

gtk\_window\_set\_title (GTK\_WINDOW (window), title);

gtk\_container\_set\_border\_width (GTK\_CONTAINER (window), 0);

vbox = gtk\_vbox\_new (FALSE, 10);

gtk\_container\_set\_border\_width (GTK\_CONTAINER (vbox), 10);

gtk\_container\_add (GTK\_CONTAINER (window), vbox);

gtk\_widget\_show (vbox);

sprintf (id\_char, "本进程 ID:%d", getpid ());

label = gtk\_label\_new (id\_char);

gtk\_box\_pack\_start (GTK\_BOX (vbox), label, FALSE, FALSE, 0);

gtk\_widget\_show (label);

sprintf (id\_char, "父进程 ID:%d", getppid ());

label = gtk\_label\_new (id\_char);

gtk\_box\_pack\_start (GTK\_BOX (vbox), label, FALSE, FALSE, 0);

gtk\_widget\_show (label);

pbar = gtk\_progress\_bar\_new ();

gtk\_box\_pack\_start (GTK\_BOX (vbox), pbar, FALSE, FALSE, 0);

gtk\_widget\_show (pbar);

timer = gtk\_timeout\_add (1000, progress\_timeout, pbar);

button = gtk\_button\_new\_with\_label ("close");

g\_signal\_connect\_swapped (G\_OBJECT (button), "clicked", G\_CALLBACK

(gtk\_widget\_destroy), window);

gtk\_box\_pack\_start (GTK\_BOX (vbox), button, FALSE, FALSE, 0);

GTK\_WIDGET\_SET\_FLAGS (button, GTK\_CAN\_DEFAULT);

gtk\_widget\_grab\_default (button);

gtk\_widget\_show (button);

gtk\_widget\_show (window);

gtk\_main ();

}

int main(int argc, char \*argv[])

{

int pid = fork ();

if (pid < 0)

printf ("error!\n");

else if (pid == 0)

{

int pid = fork ();

if (pid < 0)

printf ("error!\n");

else if (pid == 0)

show1 (argc,argv,"process1");

else

show2 (argc,argv,"process2"); }

else

show3 (argc,argv,"process3");

}

# 实验二 系统调用相关知识

## 实验目的

* 掌握添加系统调用的方法

## 实验内容

* 采用编译内核的方法，添加一个新的系统调用，实现文件拷贝功能编写一个应用程序，测试新加的系统调用。

## 实验设计

## 开发环境

* 主机系统版本：Window 10 64位
* 主机内存：8G
* Intel(R) Core(TM) i5-4300M CPU @2.60GHz 2.60 GHz
* 虚拟机：vmware workstation
* 虚拟机操作系统 Ubuntu16.04 64位
* 虚拟机内存：4G
* 虚拟机硬盘大小：80G
* 新添内核：LINUX 4.4.4

## 实验设计

系统调用的添加，没有太多的实验设计，主要是加强对系统调用的理解。根据我对系统调用的理解，系统调用是因为操作系统为保护自己重要部分不会被用户随便的篡写，例如时钟等部分。就把操作系统分成用户区和内核区，其中内核区基本可以调用所有的指令，但是又为了用户的方便，所以提供了端口，便于用户来使用内核区的功能。

用户如果想要从用户态进入内核态，就需要中断、异常来实现，而系统调用就是作为一种异常类型来实现的。

这次的实验，就是来修改系统内部提供给用户的指令。也就是改写一个Linux系统，在这次实验需要做好许多的先前准备，否则，在实验过程将会遇上许多的问题。

## 实验调试

## 实验步骤

#### 扩展硬盘空间

首先观察下自己虚拟机的硬盘大小，开始我的硬盘大小只有20G，显然这是不够的，于是需要将其扩展硬盘大小。个人保险起见，设置了80G的硬盘空间，如图2.1所示。



图2.1

具体操作是，先在编辑虚拟机设置处调整好所需硬盘空间大小。然后进入虚拟机内，下载好GParted（当然可以是其他的扩容工具，笔者在这里使用的是GParted），根据博客https://blog.csdn.net/Carina\_Cao/article/details/90270389内的教程进行操作，将分配的空闲空间合并到要用的地方即可。

当初卡到这里很久才得以解决，这是当时我遇到的问题，如图2.2。



图2.2

#### 下载内核源代码

个人推荐上海交通大学的网站，速度是真的快，地址是http://ftp.sjtu.edu.cn/sites/ftp.kernel.org/pub/linux/kernel/v4.x/，我使用的是4.4.4版本的内核。指导书的推荐2.6.18.1版本实在太过于久远，所以不是很推荐。

#### 修改内核文件

1. 添加源代码

第一个任务是编写加到内核中的源程序，即将要加到一个内核文件中去的一个函数，该函数的名称应该是新的系统调用名称前面加上sys\_标志。本次实验新添加两个系统调用，分别是sys\_hello和sys\_mycopy。如图2.3所示。

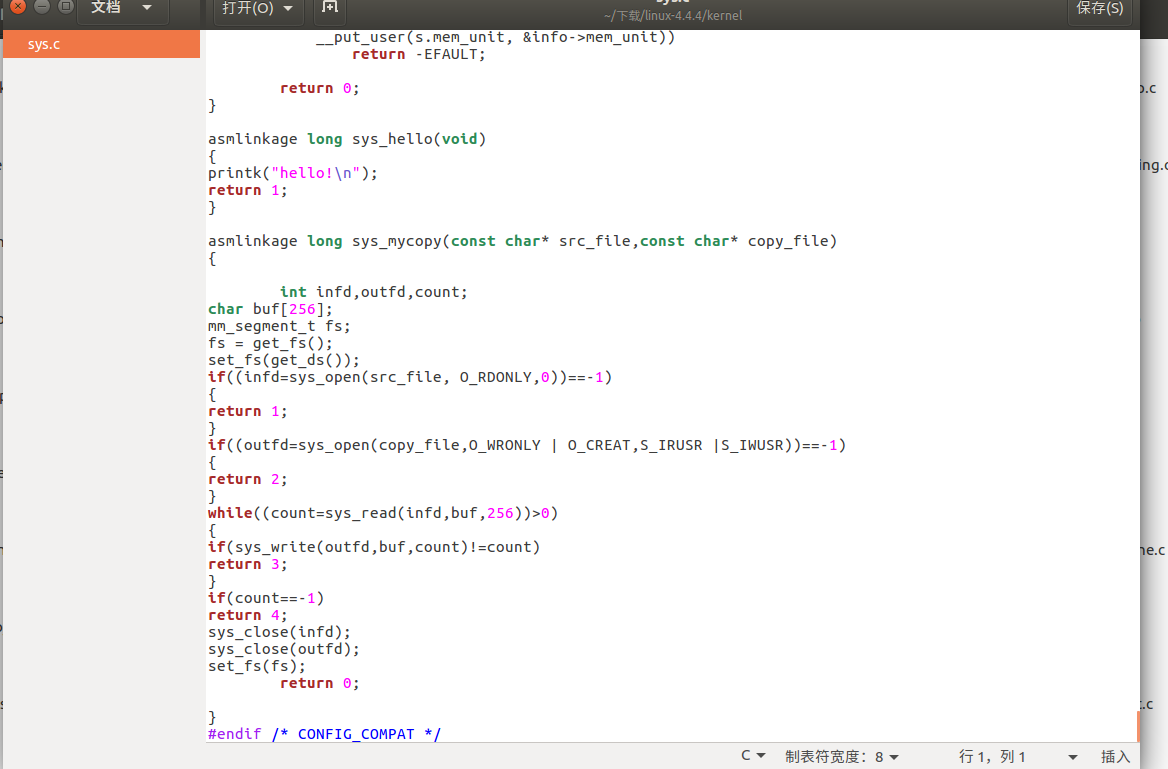


图2.3

（2）连接新的系统调用

添加新的系统调用后，下一个任务是使Linux内核的其余部分知道该程序的存在。为了从已有的内核程序中增加到新的函数的连接，需要编辑两个文件。

一个是在下载的内核中的./include/linux/syscalls.h添加函数调用，如图2.4。

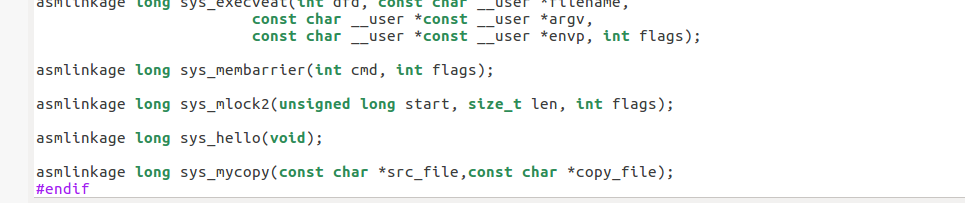


图2.4

另一个是在./arch/x86/entry/syscalls/syscall\_64.tbl添加系统调用号，接在后面继续写，所以二者分别对应调用号356和357。如图2.5。

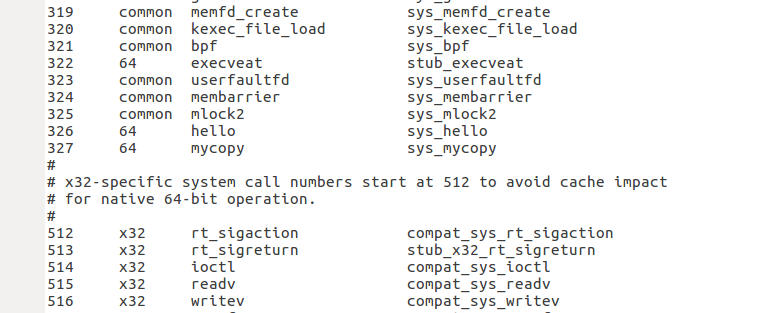


图2.5

#### 对新内核编译

可以选择先进行编译后修改的操作，对顺序并不是严格要求。我做的时候并没有考虑那么多，把编译放到后面来进行。但是，编译的时候单核进行，用了一个晚上才编译完成。以下编译的部分图片。如图2.6，图2.7，图2.8。

编译指令：

sudo menuconfig 直接选择save，然后退出即可，生成.config

sudo make 编译开始

sudo make modules\_install 安装模块

sudo make install 安装内核

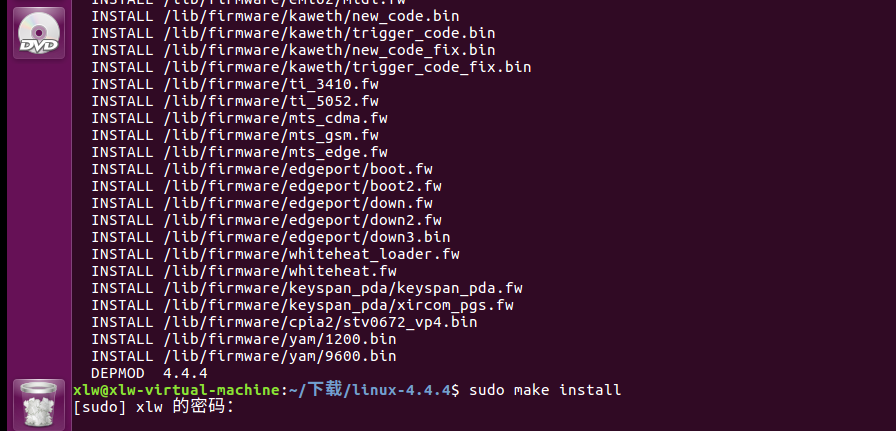


图2.6

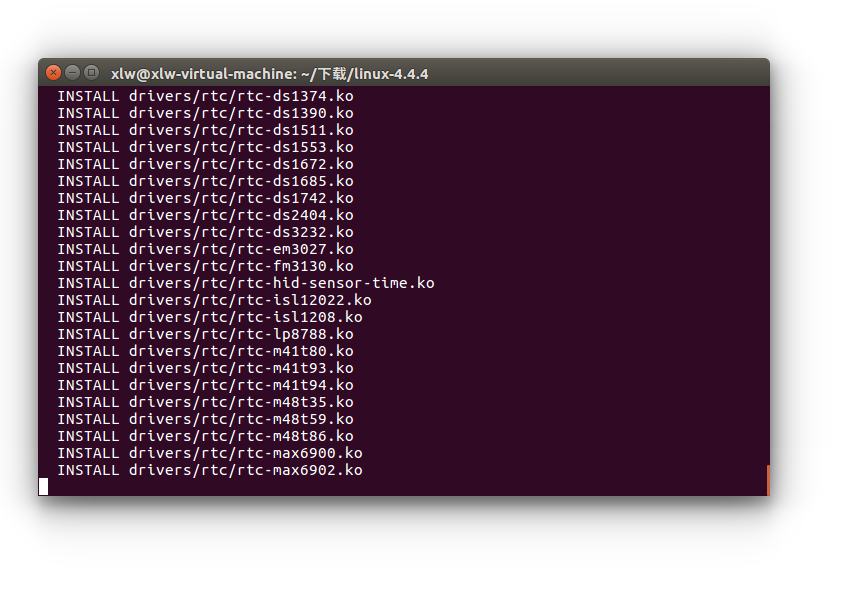


图2.7

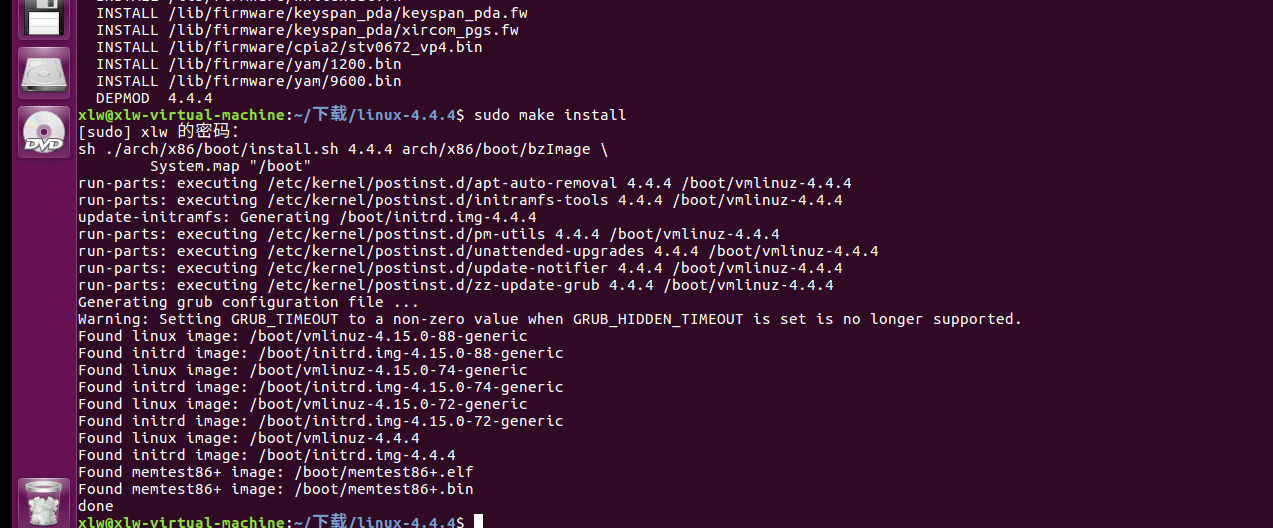


图2.8

#### 内核位置确定

内核位置一般要放到/usr/src/当中，如果不是那个位置的需要进行转移，或者在编译的时候就放到相应位置，可以有效的减少复制所浪费的时间。

由于我开始编译的时候怕出问题，所以内核位置放在其他地方，和我一样的小伙计就要注意了，这时候要把编译好的内核，/usr/src/里面。并且把生成的内核文件(arch/x86\_64/boot/bzImage)，复制进/boot中，注意保存原有内核文件

#### grub引导

编辑grub的配置文件/boot/grub/grub.cfg。然后更新配置grub的配置文件，收到这样的warning。开始以为是要不接受非零值，就把值改成了0，后面才了解那是grub的出现时间。所以尽量把时间设置到30-40即可，如图2.9。貌似我填的20。

IMG_256

图2.9

#### 重启登录

如果上面都做好了，可以直接reboot重启，然后疯狂点检shift来进入grub的界面。通过grub的高级选项，我找到我修改的核。我们选择我们选的核进入，进去调用356和357。

#### 测试代码

使用syscall来系统调用，接着测试系统调用是否添加成功。如果没有成功就回到3步骤，看代码是否有问题。由于前面已经编译过了，所以第二次编译将会很快。

## 实验调试及心得

#### 添加系统调用1

图2.10是我写调试的代码，用来调试是否能执行写好的系统调用。如图所示，直接利用syscall调用356系统号。然后编译运行的如图2.11，如果运行错误，就会返回系统的默认值-1，但这时返回的不是-1，而是我在系统写的1，说明运行成功。

接着查看系统日记，如图2.12，，因为我之前尝试过几次，所以显示了多个hello。



图2.10

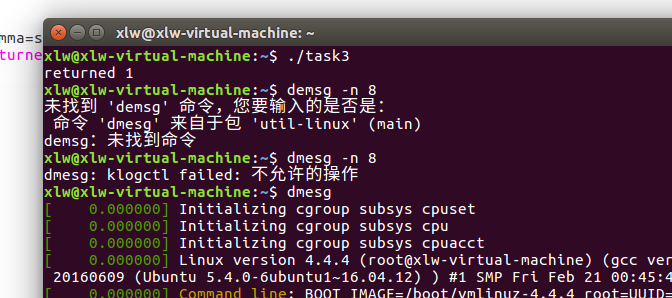


图2.11

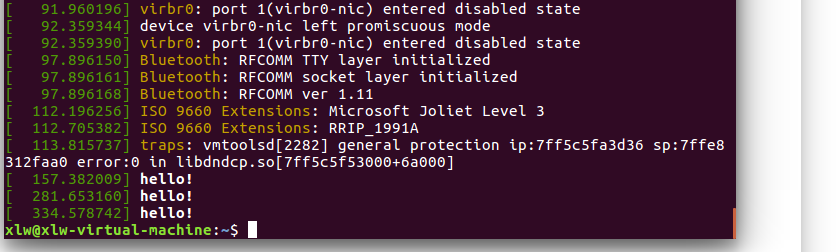


图2.12

#### 添加系统调用2

我们添加的第二个系统调用是有关文件的复制。

下面是对sys\_mycopy的测试，图2.13是代码。和实验一的复制粘贴类似，运行代码，返回值为0，说明程序成功运行，如图2.14。然后我们去看new文件是否copy成功（之前是空的），如图2.15。可以得到结果，我们成功的实现系统调用。

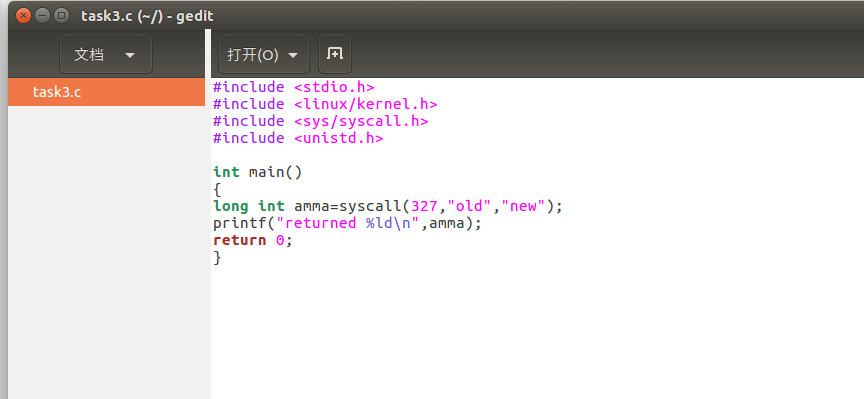


图2.13

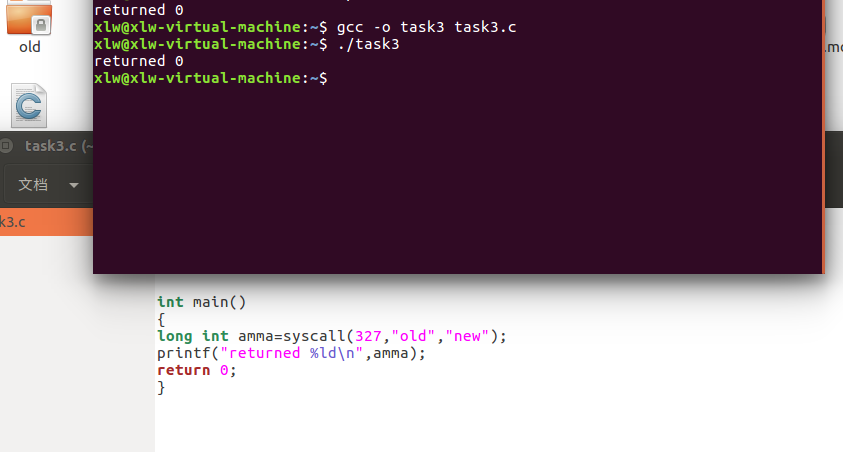


图2.14

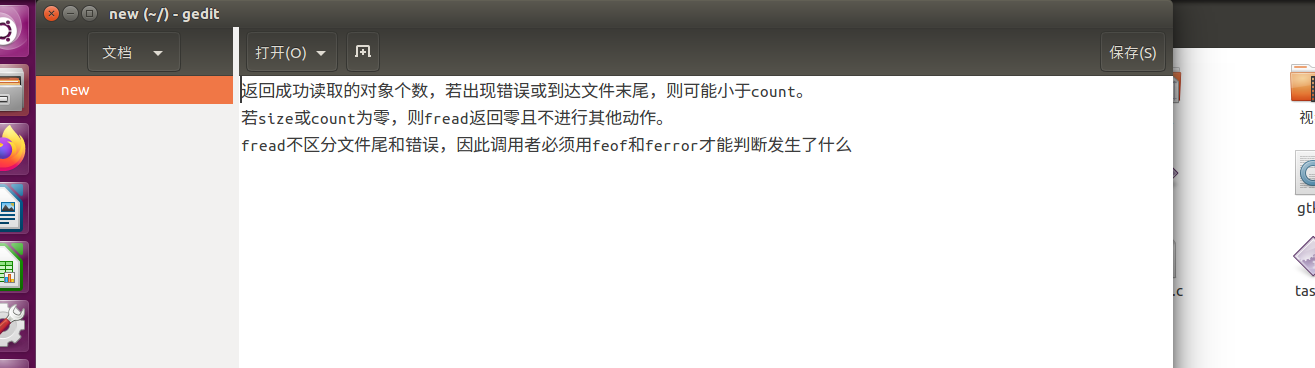


图2.15

#### 实验心得

实验有许多的注意事项需要注意，都是本人踩下的坑，需要好好地记录才行，

总结下应该有以下几点需要注意：

1. 存储空间

由于之前听到许多做的先的同学反映说，存储空间不够用导致要重来，所以大家一定要注意存储空间的使用，笔者为了谨慎，分了80G。

1. 系统下载

许多人下载的不同版本版本，有些的一直没成功，有些的一试就成功，当然我也是一试就成功的那种。但根据大家的反映，我觉得4.几版本可能相对稳定点，大家尽量找和自己原本的内核相近一点的内核最好。

1. 快照

这个及其重要，万一失败了，还有东西可以复原，而且挺快速的。（哭了，后面由于使用了autoremove把许多之前很多下好的东西全部删除了）

1. 多几个内核不丢人，由于系统编译过程需要很长的时间，而大多虚拟机初始设置的内核数是1。因此就出现了一个尴尬的现象，明明是个大工程，一个核工作，其他核休息，我足足编译一个晚上才编译过。

## 附录 实验代码

调试代码：

#include <stdio.h>

#include <linux/kernel.h>

#include <sys/syscall.h>

#include <unistd.h>

int main()

{

syscall(326);

long int amma=syscall(327,"old","new");

printf("returned %ld\n",amma);

return 0;

}

# 实验三 设备驱动相关知识

## 实验目的

* 掌握添加设备驱动程序的方法

## 实验内容

* 采用模块方法，添加一个新的字符设备驱动程序，实现打开/关闭、读/写等基本操作
* 编写一个应用程序，测试添加的驱动程序

## 实验设计

## 开发环境

* 主机系统版本：Window 10 64位
* 主机内存：8G
* Intel(R) Core(TM) i5-4300M CPU @2.60GHz 2.60 GHz
* 虚拟机：vmware workstation
* 虚拟机操作系统 Ubuntu16.04 64位
* 虚拟机内存：4G
* 虚拟机硬盘大小：80G

## 实验设计

实验过程中使用了3份代码，一份指导书内自带的设备驱动代码，一个Makefile代码，一份检测代码。

#### 设备驱动源代码

在设备驱动程序中有一个非常重要的结构file\_operations,该结构的每个域都对应着一个系统调用。用户进程利用系统调用在对设备文件进行诸如read/write操作时，系统调用通过设备文件的主设备号找到相应的设备驱动程序，然后读取这个数据结构相应的函数指针，接着把控制权交给该函数。

根据实验设计指导书的代码复制即可，具体内容：

struct file\_operations {

int (\*seek) (struct inode \* ，struct file \*， off\_t ，int);

int (\*read) (struct inode \* ，struct file \*， char ，int);

int (\*write) (struct inode \* ，struct file \*， off\_t ，int);

int (\*readdir) (struct inode \* ，struct file \*， struct dirent \* ，int);

int (\*select) (struct inode \* ，struct file \*， int ，select\_table \*);

int (\*ioctl) (struct inode \* ，struct file \*， unsined int ，unsigned long);

int (\*mmap) (struct inode \* ，struct file \*， struct vm\_area\_struct \*);

int (\*open) (struct inode \* ，struct file \*);

int (\*release) (struct inode \* ，struct file \*);

int (\*fsync) (struct inode \* ，struct file \*);

int (\*fasync) (struct inode \* ，struct file \*，int);

int (\*check\_media\_change) (struct inode \* ，struct file \*);

int (\*revalidate) (dev\_t dev);

}

编写设备驱动程序的主要工作是编写子函数，并填充file\_operations的各个域。

例如：

Struct file\_operations my\_fops={

.read=my\_read,

.write=my\_write,

.open=my\_open,

.release=my\_release

}

然后再定义函数my\_read,my\_write,my\_open,my\_release相应的函数体。

例如：

static ssize\_t my\_open(struct inode \*inode,struct file \*file){

static int counter=0;

if(Device\_Open)

return -EBUSY;

Device\_Open++;

/\*写入设备的信息\*/

sprintf(msg,"the device has been called %d times\n",counter++);

msg\_ptr=msg;

return 0;

}

同时对于可卸载的内核模块（LKM）,至少还有两个基本的模块：

例如本例中的：

static int \_\_init my\_init(void){

int result;

result=register\_chrdev(0,"sky\_driver",&my\_fops);

if(result<0){

printk("error:can not register the device\n");

return -1;

}

if(my\_major==0){

my\_major=result;

printk("<1>hehe,the device has been registered!\n");

printk("<1>the virtual device was assigned major number %d.\n",my\_major);

printk("<1>To talk to the driver,create a dev file with\n");

printk("<1>'mknod/dev/my c %d 0'\n",my\_major);

printk("<1>Remove the dev and the file when done\n");

}

return 0;

}

static void \_\_exit my\_exit(void){

unregister\_chrdev(my\_major,"sky\_driver");

printk("<1>unloading the device\n");

}

my\_init 用于注册设备，获得设备的主设备号

调用register\_chrdev(0,“sky\_driver(设备名)”，&my\_fops);

my\_exit 用于注销设备

调用unregister\_chrdev(my\_major, “sky\_driver(设备名)”);

然后在程序尾再调用这两个函数

Module\_init(my\_init);

Module\_exit(my\_exit)

MODULE\_LICENSE(“GPL”);

#### Makefile文件

修改Makefile,只要一句即可:obj-m +=sky\_driver.o,但实际上我用的是devDrv.o,如图3.1所示

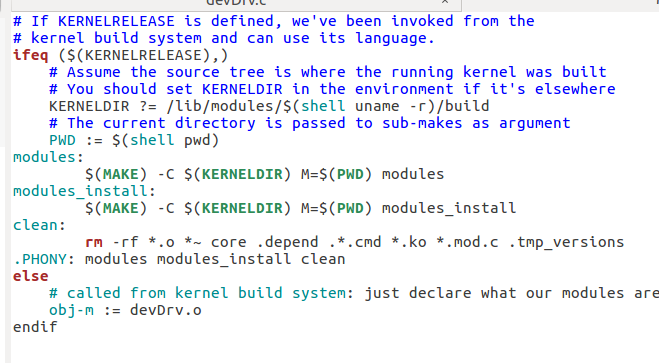


图3.1

#### 测试代码

主要思路就是先读取所要使用的设备号，如果能找到该文件，就开始进行读写操作，测试是否成功。代码如图3.2所示。

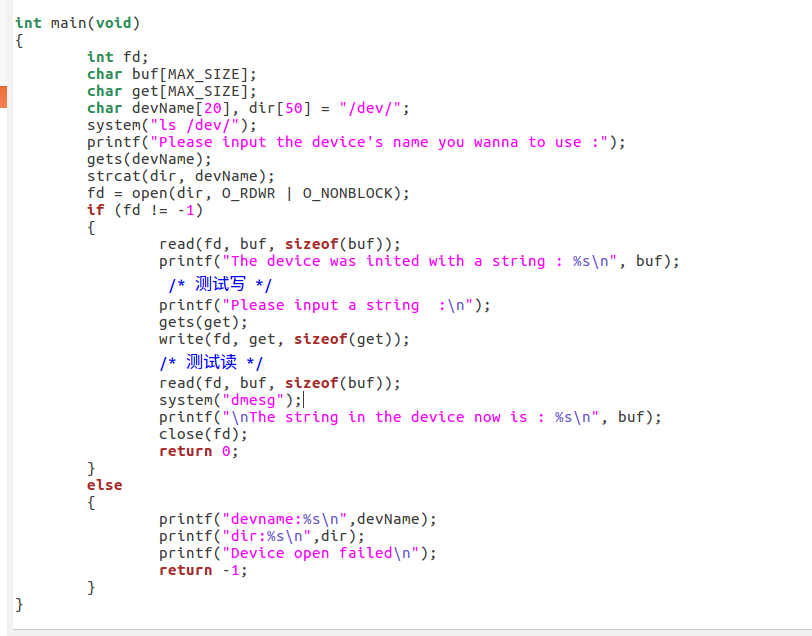


图3.2

## 实验调试

## 实验步骤

#### 编写设备驱动代码

在实验设计中已经完成，可以直接进行下一步,和指导书上有点区别的是我的设备名是myDevice，模块名是devDrv。这个其实随便进行调整的，所以只是单独列出而已。

#### 编译代码

由于编写好了Makefile文件，因此只需要在当前文件目录打开终端输入make即可，如果编译成功将得到.ko文件。

值的注意的是：此步编译过程中必须没有错误或者任何的警告，否则必须对错误和警告信息提示的位置进行修改，然后重新编译，直到没有任何错误或者警告（warning）为止。

#### 安装模块

命令：insmod devDrv.ko，需要注意的是，必须要用sudo命令，否则就会没权限，正如图3.3所示。

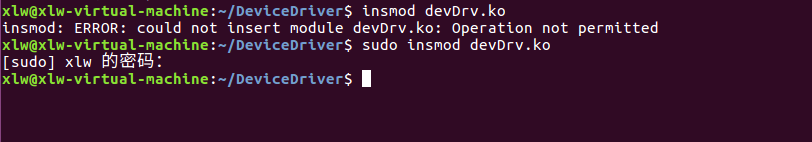


图3.3

#### 模块查询

确认模块是否已经安装完毕，此时 cat /proc/devices或dmesg会看到在字符设备中有247 myDevice,如图3.4，图3.5。前面的是系统分配的主设备号，后面是设备注册名。这说明我们加载成功。

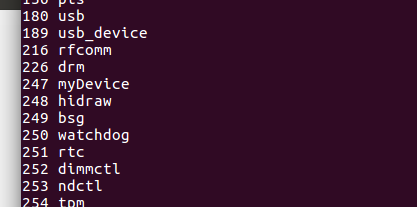


图3.4

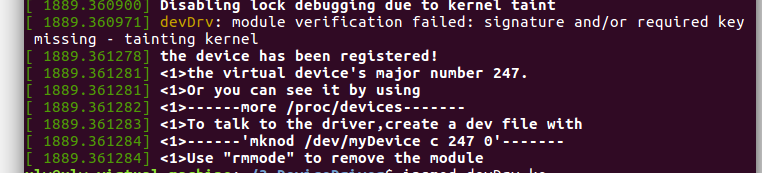


图3.5

#### 创建新的虚拟设备文件

命令：sudo mknod /dev/myDevice c 247 0

在此命令中，第一个参数是新建设备文件的地址和名字，第二个参数是指创建的是字符设备文件，第三个参数是主设备号，第四个参数是从设备号。记住要加sudo，否则权限不够。如图3.6。

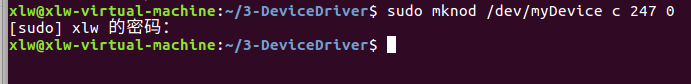


图3.6

执行成功会在/dev中看到一个新的设备文件myDevice 。输入命令ls /dev/，如图3.7。这时设备驱动可以说是彻底完成。



图3.7

#### 测试新的设备驱动

准备在设计时候写的测试代码，进行测试。需要注意的是，我们需要超级用户的权限，因此执行su命令，进入超级用户。或者sudo执行该程序也可以，否则会说这个设备打不开。

如图3.8，我打开了设备，并且输入一串字符串，然后查看设备内的信息，以及系统信息如图3.9。根据图上的信息，可以说明设备驱动成功。

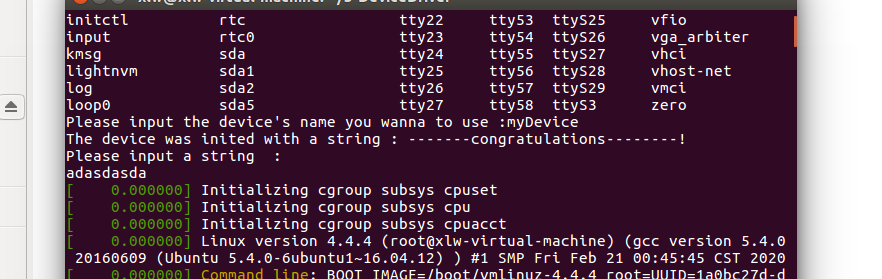


图3.8

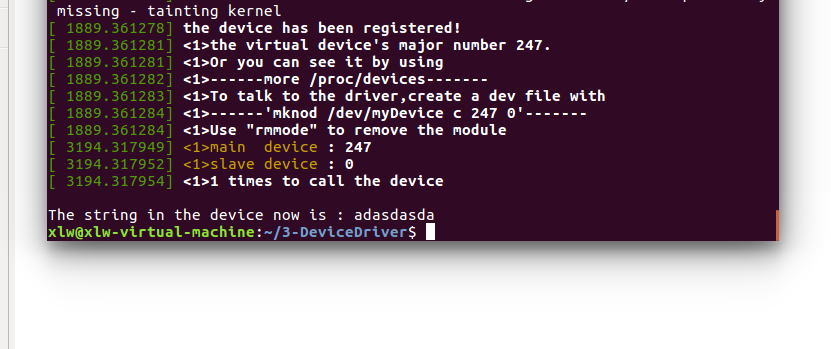


图3.9

#### 卸载操作

删除新增的字符设备文件

命令：rm /dev/myDevice

删除模块

命令：rmmod devDrv

## 实验调试及心得

#### 实验调试

测试新添加的设备驱动已经在3.4.1.6中完成，所以不再赘述。

#### 实验心得

本次实验基本上就是按照实验指导书上一步一步的做过来的，然后做的过程都挺顺利的，但是在最后测试的时候需要使用超级用户命令，否则设备驱动将无法运行，曾经在这个错误上找了很久的解决措施。

## 附录 实验代码

Devdrv.c代码：

#include "linux/kernel.h"

#include "linux/module.h"

#include "linux/fs.h"

#include "linux/init.h"

#include "linux/types.h"

#include "linux/errno.h"

#include "linux/uaccess.h"

#include "linux/kdev\_t.h"

#define MAX\_SIZE 1024

static int my\_open(struct inode \*inode, struct file \*file);

static int my\_release(struct inode \*inode, struct file \*file);

static ssize\_t my\_read(struct file \*file, char \_\_user \*user, size\_t t, loff\_t \*f);

static ssize\_t my\_write(struct file \*file, const char \_\_user \*user, size\_t t, loff\_t \*f);

static char message[MAX\_SIZE] = "-------congratulations--------!";

static int device\_num = 0;//设备号

static int counter = 0;//计数用

static int mutex = 0;//互斥用

static char\* devName = "myDevice";//设备名

struct file\_operations pStruct =

{ open:my\_open, release:my\_release, read:my\_read, write:my\_write, };

/\* 注册模块 \*/

int init\_module()

{

int ret;

/\* 函数中第一个参数是告诉系统，新注册的设备的主设备号由系统分配，

\* 第二个参数是新设备注册时的设备名字，

\* 第三个参数是指向file\_operations的指针，

\* 当用设备号为0创建时，系统一个可以用的设备号创建模块 \*/

ret = register\_chrdev(0, devName, &pStruct);

if (ret < 0)

{

printk("regist failure!\n");

return -1;

}

else

{

printk("the device has been registered!\n");

device\_num = ret;

printk("<1>the virtual device's major number %d.\n", device\_num);

printk("<1>Or you can see it by using\n");

printk("<1>------more /proc/devices-------\n");

printk("<1>To talk to the driver,create a dev file with\n");

printk("<1>------'mknod /dev/myDevice c %d 0'-------\n", device\_num);

printk("<1>Use \"rmmode\" to remove the module\n");

return 0;

}

}

/\* 注销模块，函数名很特殊 \*/

void cleanup\_module()

{

unregister\_chrdev(device\_num, devName);

printk("unregister it success!\n");

}

static int my\_open(struct inode \*inode, struct file \*file)

{

if(mutex)

return -EBUSY;

mutex = 1;//上锁

printk("<1>main device : %d\n", MAJOR(inode->i\_rdev));

printk("<1>slave device : %d\n", MINOR(inode->i\_rdev));

printk("<1>%d times to call the device\n", ++counter);

try\_module\_get(THIS\_MODULE);

return 0;

}

/\* 每次使用完后会release \*/

static int my\_release(struct inode \*inode, struct file \*file)

{

printk("Device released!\n");

module\_put(THIS\_MODULE);

mutex = 0;//开锁

return 0;

}

static ssize\_t my\_read(struct file \*file, char \_\_user \*user, size\_t t, loff\_t \*f)

{

if(copy\_to\_user(user,message,sizeof(message)))

{

return -EFAULT;

}

return sizeof(message);

}

static ssize\_t my\_write(struct file \*file, const char \_\_user \*user, size\_t t, loff\_t \*f)

{

if(copy\_from\_user(message,user,sizeof(message)))

{

return -EFAULT;

}

return sizeof(message);

}

Test.c代码：

#include <sys/types.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#define MAX\_SIZE 1024

int main(void)

{

int fd;

char buf[MAX\_SIZE];

char get[MAX\_SIZE];

char devName[20], dir[50] = "/dev/";

system("ls /dev/");

printf("Please input the device's name you wanna to use :");

gets(devName);

strcat(dir, devName);

fd = open(dir, O\_RDWR | O\_NONBLOCK);

if (fd != -1)

{

read(fd, buf, sizeof(buf));

printf("The device was inited with a string : %s\n", buf);

/\* 测试写 \*/

printf("Please input a string :\n");

gets(get);

write(fd, get, sizeof(get));

/\* 测试读 \*/

read(fd, buf, sizeof(buf));

system("dmesg");

printf("\nThe string in the device now is : %s\n", buf);

close(fd);

return 0;

}

else

{

printf("devname:%s\n",devName);

printf("dir:%s\n",dir);

printf("Device open failed\n");

return -1;

}

}

Makefile代码：

# If KERNELRELEASE is defined, we've been invoked from the

# kernel build system and can use its language.

ifeq ($(KERNELRELEASE),)

# Assume the source tree is where the running kernel was built

# You should set KERNELDIR in the environment if it's elsewhere

KERNELDIR ?= /lib/modules/$(shell uname -r)/build

# The current directory is passed to sub-makes as argument

PWD := $(shell pwd)

modules:

$(MAKE) -C $(KERNELDIR) M=$(PWD) modules

modules\_install:

$(MAKE) -C $(KERNELDIR) M=$(PWD) modules\_install

clean:

rm -rf \*.o \*~ core .depend .\*.cmd \*.ko \*.mod.c .tmp\_versions

.PHONY: modules modules\_install clean

else

# called from kernel build system: just declare what our modules are

obj-m := devDrv.o

endif

# 实验四 /proc文件相关知识

## 实验目的

* 使用GTK/QT实现一个系统监控器（选做）

## 实验内容

* 了解/proc文件的特点和使用方法
* 监控系统状态，显示系统部件的使用情况
* 用图形界面监控系统状态，包括系统基本信息、CPU和内存利用率、所有进程信息等(可自己补充、添加其他功能)

## 实验设计

## 开发环境

* 主机系统版本：Window 10 64位
* 主机内存：8G
* Intel(R) Core(TM) i5-4300M CPU @2.60GHz 2.60 GHz
* 虚拟机：vmware workstation
* 虚拟机操作系统 Ubuntu16.04 64位
* 虚拟机内存：4G
* 虚拟机硬盘大小：80G
* 图形界面：Gtk

## 实验设计

用户和应用程序可以通过/proc得到系统的信息，并可以改变内核的某些参数。由于系统的信息是动态改变的，所以用户或应用程序读取proc文件时，proc文件系统是动态从系统内核读出所需信息并提交的。

要显示系统信息，只需进行相应的文件操作。首先打开相应的文件，读取所需要的信息，将其写入一个缓冲区中，然后将缓冲区的内容加到Gtk的相应的控件上面去，最后将控件组合显示即可。

#### CPU信息

参考了设计指导书以及网上csdn博客的介绍，总结下proc信息的读取。

首先我可以先看一下/proc里关于CPU使用的信息，使用cat /proc/cpuinfo你可以查看系统的内存信息，或者直接打开该文件，打开后的文件如图4.1所示。

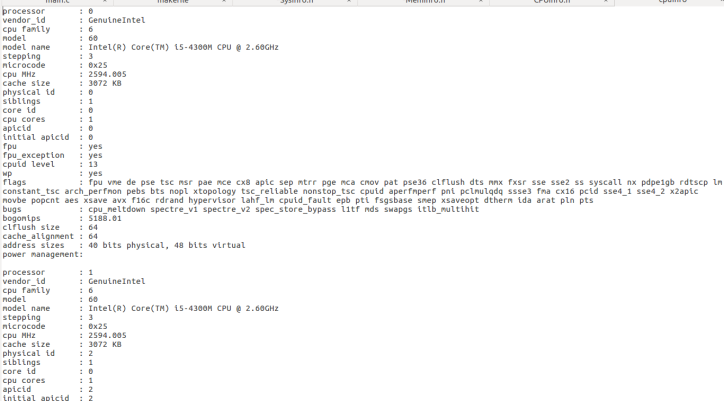


图4.1

由上图可知所需的信息所在文件中的位置，然后将其写入缓冲区，加载到gtk相应的控件上。

具体的实现过程可以直接调用getinfo函数获取CPU的信息，把其写入缓冲区去，然后把缓冲区的内容读取出来填在gtk的控件上，这样子做可以帮助我们排版。实现的代码如下所示：

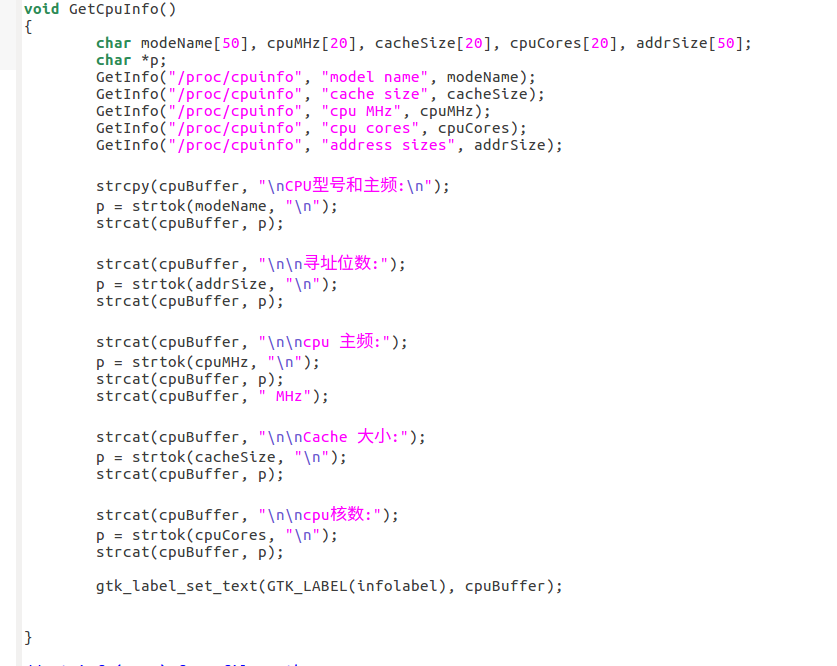


图4.2

除了基本信息的打印，还可以通过上面的数据，进行CP时间的计算和CPU利用率的计算。

其中CPU时间可以通过以下等式计算：

CPU(total) =user + nice + system +idle + iowait

计算CPU利用率时，采样两个足够短的时间，分别为t1,t2。则CPU利用率可以用下等式计算：

pcpu = 100 \*(total – idle) / total

total =total(t2) – total(t1)

idle =idle(t2) – idle(t1)

由于CPU利用率随时间改变，因此对于显示出的CPU利用率，需要随时更新，我设置为1秒更新一次。

#### 操作系统信息

同样的道理，不过有关操作系统的信息位于/proc/version里面，系统信息查看模块比较简单，不需要更新，因此设置一个label显示该信息即可

#### 内存信息

相对于CPU来说，内存部分的信息较为简单，上表中内存的信息在文件”/proc/meminfo”中对于的字段即可以寻找到。此外还需要计算内存和交换分区的使用率，内存使用率按照以下公式计算：

pmm= (MemTotal – MemFree) / MemTotal

pswap= (SwapTotal – SwapFree) / SwapTotal

       由于内存和交换分区使用率也随着时间而变化，因此上述两个值得信息我也设计每秒更新一次。

#### 进程信息

进程信息模块主要是获得进程的详细信息，同时，我设计了刷新功能、用PID搜索进程详细信息功能以及杀死进行的功能。

 Linux中，每一个进程在目录”/proc/”下对应一个文件夹，该文件夹名称为进程号，因此符合该条件的文件夹的数量即为进程数，对于每一个进程来说，其信息在”/proc/<pid>/”路径下。本次的信息主要用到了两个文件，分别为：”/proc/<pid>/status”和”/proc/<pid>/stat”

为了获得进程的优先级，需要查看stat文件，在stat文件中也保存着关于该进程的一些详细信息，包括进程名，运行状态，PID等，这些信息已在status中获得，本次主要获得优先级信息即可。priority在该文件的第18位中。

通过上述过程即可获得关于进程所需要的信息，此外，还需要有通过PID查看进程、更新进程信息以及杀死进程的功能。为此，设计了一个GtkEntry，在其中输入进程号即可以进行查询，为了方便用户使用，当点击某一进程的时候，进程号会自动填写到entry中。

进程的搜索是进入”/proc/<pid>/”这个目录，如果该目录不存在则输出提示信息，然后再根据上述过程弹出对话框展示此进程详细信息。

杀死进程需要调用系统命令：system(“kill <pid>”)，该功能的实现较为简单，不过杀死进程的过程中会遇到权限不够的问题，因此最好在杀死进程时进行权限的判断。

## 实验调试

## 实验调试及心得

#### 实验调试

下图是对系统信息的打印

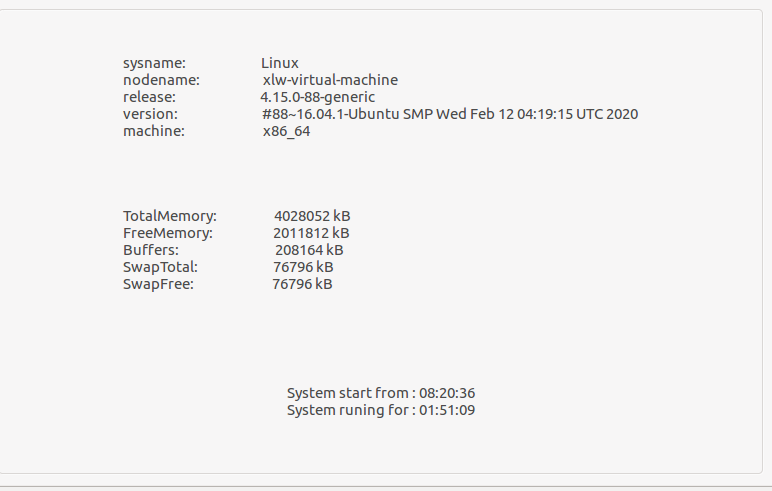


图4.3

进程的查看：

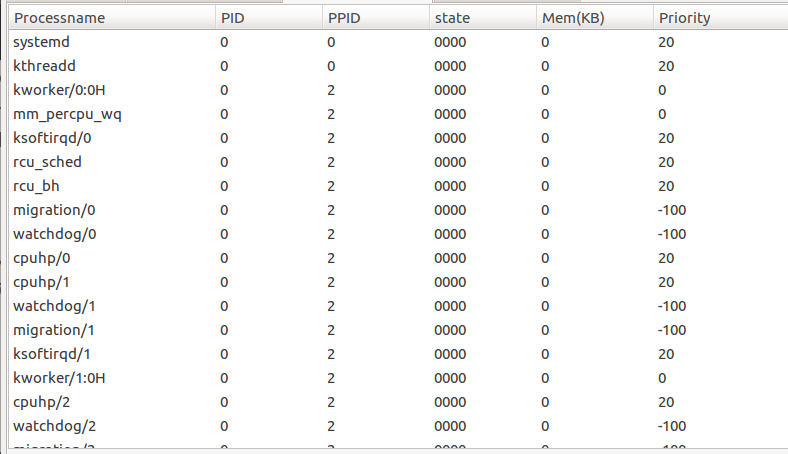


图4.4

Cpu的查看：

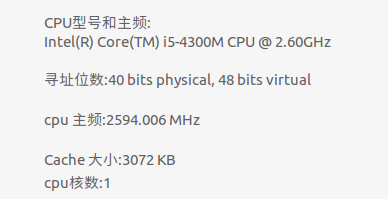


图4.5

#### 实验心得

本次实验只是简单的利用gtk来对Linux操作系统信息的打印，可以实现CPU利用率的计算、操作系统信息的查询、进程控制等功能，实验过程出现了小问题，当时还不会gdb调试，因此就就进行手动下断点的蠢方法，过程有点艰辛，但是结果是美好的，我也在此次实验过程中学习到gtk的进阶使用，查阅了相当多的博客才学会。

## 附录 实验代码

Main.c

#include <sys/stat.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <fcntl.h>

#include <dirent.h>

#include <gdk/gdkkeysyms.h>

#include <gtk/gtk.h>

#include <gdk/gdk.h>

#include <cairo.h>

#include <glib.h>

#include <time.h>

#include "CPUinfo.h"

#include "Meminfo.h"

#include "Processinfo.h"

#include "Sysinfo.h"

void CreateVersion(GtkWidget \*vbox);

void CreatePageFour(GtkWidget \*notebook);

gint refresh\_clock(gpointer data);

GtkWidget \*statusbar;

int main(int argc, char\* argv[])

{

int i=0;

GtkWidget \*window, \*main\_vbox;

gtk\_init(&argc, &argv);

//printf("%d\n",i++);//0

window = gtk\_window\_new(GTK\_WINDOW\_TOPLEVEL);

gtk\_window\_set\_title(GTK\_WINDOW(window), "System monitor");

//printf("%d\n",i++);//1

gtk\_window\_set\_default\_size(GTK\_WINDOW(window), 800, 600);

//printf("%d\n",i++);//2

gtk\_window\_set\_position(GTK\_WINDOW(window), GTK\_WIN\_POS\_CENTER);

//printf("%d\n",i++);//3

gtk\_widget\_show(window);

//printf("%d\n",i++);//4

g\_signal\_connect(G\_OBJECT(window),"destroy",G\_CALLBACK(gtk\_main\_quit),NULL);

main\_vbox = gtk\_vbox\_new(FALSE, 10);

gtk\_container\_set\_border\_width(GTK\_CONTAINER(main\_vbox), 5);

//printf("%d\n",i++);//5

gtk\_container\_add(GTK\_CONTAINER(window), main\_vbox);

//printf("%d\n",i++);//6

gtk\_widget\_show(main\_vbox);

//printf("%d\n",i++);//7

//printf("%d\n",i++);//8(有问题，解决)

CreateVersion(main\_vbox);

//printf("%d\n",i++);//9

gtk\_timeout\_add(500, refresh\_cpu, NULL);

//printf("%d\n",i++);//10

gtk\_timeout\_add(500, refresh\_mm, NULL);

//printf("%d\n",i++);//11

gtk\_timeout\_add(500, refresh\_swap, NULL);

//printf("%d\n",i++);//13

gtk\_widget\_show\_all(window);

//printf("%d\n",i++);//14

gtk\_main();

}

//Create the main version window

void CreateVersion(GtkWidget \*vbox)

{

GtkWidget \*notebook;

gint context\_id;

notebook = gtk\_notebook\_new();

gtk\_notebook\_set\_tab\_pos((GtkNotebook \*)notebook, GTK\_POS\_TOP);

gtk\_notebook\_set\_show\_border((GtkNotebook \*) notebook, FALSE);

gtk\_box\_pack\_start(GTK\_BOX(vbox), notebook, TRUE, TRUE, 0);

gtk\_widget\_show(notebook);

statusbar = gtk\_statusbar\_new();

gtk\_box\_pack\_start(GTK\_BOX(vbox), statusbar, TRUE, TRUE, 0);

gtk\_widget\_show(statusbar);

context\_id = gtk\_statusbar\_get\_context\_id(GTK\_STATUSBAR(statusbar), "Timestatus");

gtk\_timeout\_add(1000, refresh\_clock, GINT\_TO\_POINTER(context\_id));

refresh\_clock(GINT\_TO\_POINTER(context\_id));

CreatePageOne(notebook);

CreatePageTwo(notebook);

CreatePageThree(notebook);

CreatePageFive(notebook);

gtk\_notebook\_set\_current\_page(GTK\_NOTEBOOK(notebook), 1);

}

void CreatePageFour(GtkWidget \*notebook)

{

GtkWidget \*label, \*vbox, \*frame;

frame = gtk\_frame\_new("");

gtk\_container\_set\_border\_width(GTK\_CONTAINER(frame), 10);

gtk\_widget\_show(frame);

gtk\_notebook\_append\_page(GTK\_NOTEBOOK(notebook), frame, gtk\_label\_new("Page4"));

}

gint refresh\_clock(gpointer data)

{

time\_t t;

t=time(NULL);

char msg[1000], temp[100];

char \*buf, \*p;

buf = ctime(&t);

p = strtok(buf, "\n");

//使用率的计算

strcpy(msg, buf);

sprintf(temp, " CPU used: %2.2lf%%", cpu\_rate \* 100);

strcat(msg, temp);

sprintf(temp, " Memory used: %2.2lf%%", mm\_rate \* 100);

strcat(msg, temp);

sprintf(temp, " Swap used %2.6lf%%", swap\_rate \* 100);

strcat(msg, temp);

//设置到每一页的最底部

gtk\_statusbar\_push(GTK\_STATUSBAR(statusbar), GPOINTER\_TO\_INT(data), msg);

}

CPUinfo.h

#ifndef \_\_CPUINFO\_H

#define \_\_CPUINFO\_H

void CreatePageOne(GtkWidget \*notebook); //CPU

void DrawingCPU(void);

gint refresh\_cpu(gpointer data);

static gboolean cpu\_expose\_event(GtkWidget \*widget, GdkEventExpose \*event, gpointer data);

static gboolean cpu\_configure\_event(GtkWidget \*widget, GdkEventConfigure \*event, gpointer data);

float getCpuUseRatio(int n);

void GetCpuInfo();

void GetInfo(char \*path, char \*name, char info[]);

GtkWidget \*draw\_cpu;

GtkWidget \*window;

GtkWidget \*labelofcpu; //cpu rate

GtkWidget \*infolabel; //cpu infomation

GdkPixmap \*cpu\_graph;

/\* 全局变量 \*/

static gint cpuPoints[100];

static gfloat cpu\_rate = 0.0;

float zuser = 0, ztotal = 0, zidle = 0;

char cpuBuffer[2000];

static gboolean cpu\_expose\_event(GtkWidget \*widget, GdkEventExpose \*event, gpointer data)

{

gdk\_draw\_drawable(widget->window, widget->style->fg\_gc[GTK\_WIDGET\_STATE(

widget)], cpu\_graph, event->area.x, event->area.y, event->area.x,

event->area.y, event->area.width, event->area.height);

return FALSE;

}

static gboolean cpu\_configure\_event(GtkWidget \*widget, GdkEventConfigure \*event, gpointer data)

{

if (cpu\_graph)

{

g\_object\_unref(cpu\_graph);

}

cpu\_graph = gdk\_pixmap\_new(widget->window, widget->allocation.width,

widget->allocation.height, -1);

gdk\_draw\_rectangle(cpu\_graph, widget->style->white\_gc, TRUE, 0, 0,

widget->allocation.width, widget->allocation.height);

return TRUE;

}

/\* 取得cpu利用率 \*/

float getCpuUseRatio(int n)//1:user 2:sys 3:iowait

{

FILE \*fp;

char buffer[1024];

size\_t buf;

float useRatio;

float sysRatio;

float iowaitRatio;

/\* 分别为用户模式，低优先级用户模式，内核模式，空闲的处理器时间 \*/

float user = 0, nice = 0, sys = 0, idle = 0, iowait = 0;

fp = fopen("/proc/stat", "r");

/\* 从fp中读取sizeof(buffer)块数据放入buffer，每次读一块 \*/

buf = fread(buffer, 1, sizeof(buffer), fp);

fclose(fp);

if (buf == 0)

return 0;

buffer[buf] == '\0';

sscanf(buffer, "cpu %f %f %f %f %f", &user, &nice, &sys, &idle, &iowait);

if (idle <= 0)

idle = 0;

//useRatio = 100 \* (user - zuser) / (user + nice + sys + idle - ztotal);

useRatio = 100 \* (user - zuser) / (idle - zidle);

sysRatio = 100 \* sys / user;

iowaitRatio = 1000 \* iowait / idle;

if (useRatio > 100)

useRatio = 100;

ztotal = user + nice + sys + idle;

zuser = user;

zidle = idle;

cpu\_rate = useRatio;

if(n == 1)

return useRatio;

else if(n == 2)

return sysRatio;

else if(n == 3)

return iowaitRatio;

}

void CreatePageOne(GtkWidget \*notebook) //CPU

{

GtkWidget \*vbox, \*frame, \*CPUframe, \*infoframe;

frame = gtk\_frame\_new("");

gtk\_container\_set\_border\_width(GTK\_CONTAINER(frame), 10);

gtk\_widget\_show(frame);

gtk\_notebook\_append\_page(GTK\_NOTEBOOK(notebook), frame, gtk\_label\_new("CPU infomation"));

GtkWidget \*table = gtk\_table\_new(10, 10, TRUE);

gtk\_container\_add(GTK\_CONTAINER(frame), table);

CPUframe = gtk\_frame\_new("CPU使用曲线");

gtk\_table\_attach\_defaults(GTK\_TABLE(table), CPUframe, 1, 9, 0, 5);

gtk\_widget\_show(CPUframe);

infoframe = gtk\_frame\_new("CPU information");

gtk\_table\_attach\_defaults(GTK\_TABLE(table), infoframe, 0, 10, 5, 10);

gtk\_widget\_show(infoframe);

infolabel = gtk\_label\_new("ddd");

gtk\_container\_add(GTK\_CONTAINER(infoframe), infolabel);

gtk\_widget\_show(infolabel);

GetCpuInfo();

gtk\_widget\_show(table);

draw\_cpu = gtk\_drawing\_area\_new();

gtk\_widget\_set\_app\_paintable(draw\_cpu, TRUE);

gtk\_drawing\_area\_size(GTK\_DRAWING\_AREA(draw\_cpu), 40, 80);

gtk\_container\_add(GTK\_CONTAINER(CPUframe), draw\_cpu);

gtk\_widget\_show(draw\_cpu);

DrawingCPU();

}

void DrawingCPU()

{

int i;

for (i = 0; i < 100; i++)

{

/\* 要取[a,b)之间的随机整数（包括a，但不包括b)，使用：

\* (rand() % (b - a)) + a \*/

cpuPoints[i] = 999;

}

g\_signal\_connect(draw\_cpu, "expose\_event",G\_CALLBACK(cpu\_expose\_event), NULL);

g\_signal\_connect(draw\_cpu, "configure\_event", G\_CALLBACK(cpu\_configure\_event), NULL);

gtk\_widget\_show(draw\_cpu);}

gint refresh\_cpu(gpointer data)

{

/\* 建矩形绘图区 \*/

GdkGC \*gc\_chart\_cpu = gdk\_gc\_new(draw\_cpu->window);

/\* 背景颜色 \*/

GdkColor color;

color.red = 0xFFFF;

color.green = 0xFFFF;

color.blue = 0xFFFF;

gdk\_gc\_set\_rgb\_fg\_color(gc\_chart\_cpu, &color);

int width, height, curPoint, step;

cpu\_rate = getCpuUseRatio(1) / 100;

gdk\_draw\_rectangle(cpu\_graph, gc\_chart\_cpu, TRUE, 0, 0,draw\_cpu->allocation.width, draw\_cpu->allocation.height);

width = draw\_cpu->allocation.width;

height = draw\_cpu->allocation.height;

curPoint = (int) (cpu\_rate \* (double) height);

cpuPoints[99] = height - curPoint;

int i;

for (i = 0; i < 99; i++)

cpuPoints[i] = cpuPoints[i + 1];

step = width / 99;

GdkGC \*gc = gdk\_gc\_new(GDK\_DRAWABLE(cpu\_graph));

gdk\_color\_parse("black", &color);

gdk\_gc\_set\_foreground(gc, &color);

gdk\_gc\_set\_line\_attributes(gc, 1, GDK\_LINE\_SOLID, GDK\_CAP\_ROUND,GDK\_JOIN\_MITER);

for (i = 99; i >= 1; i--)

gdk\_draw\_line(GDK\_DRAWABLE(cpu\_graph), gc, i \* step, cpuPoints[i], (i - 1) \* step, cpuPoints[i - 1]);

gtk\_widget\_queue\_draw(draw\_cpu);

GetCpuInfo();

return TRUE;

}

//页面一，CPU的信息

void GetCpuInfo()

{

char modeName[50], cpuMHz[20], cacheSize[20], cpuCores[20], addrSize[50];

char \*p;

GetInfo("/proc/cpuinfo", "model name", modeName);

GetInfo("/proc/cpuinfo", "cache size", cacheSize);

GetInfo("/proc/cpuinfo", "cpu MHz", cpuMHz);

GetInfo("/proc/cpuinfo", "cpu cores", cpuCores);

GetInfo("/proc/cpuinfo", "address sizes", addrSize);

strcpy(cpuBuffer, "\nCPU型号和主频:\n");

p = strtok(modeName, "\n");

strcat(cpuBuffer, p);

strcat(cpuBuffer, "\n\n寻址位数:");

p = strtok(addrSize, "\n");

strcat(cpuBuffer, p);

strcat(cpuBuffer, "\n\ncpu 主频:");

p = strtok(cpuMHz, "\n");

strcat(cpuBuffer, p);

strcat(cpuBuffer, " MHz");

strcat(cpuBuffer, "\n\nCache 大小:");

p = strtok(cacheSize, "\n");

strcat(cpuBuffer, p);

strcat(cpuBuffer, "\n\ncpu核数:");

p = strtok(cpuCores, "\n");

strcat(cpuBuffer, p);

gtk\_label\_set\_text(GTK\_LABEL(infolabel), cpuBuffer);

}

void GetInfo(char \*path, char \*name, char info[])

{

int fd = open(path, O\_RDONLY);

char temp[1000];

int k = 0;

read(fd, temp, sizeof(temp));

close(fd);

char \*p = NULL;

p = strstr(temp, name);

while(\*p != ':')

p++;

//p = p + 2;

p++;

while(\*p == ' ')

p++;

while(\*p != '\n')

{

info[k] = \*p;

p++;

k++;

}

info[k] = '\n';

info[k+1] = '\0';

}

#endif

Meninfo.h代码：

//实现曲线图

#ifndef \_\_MEMINFO\_H

#define \_\_MEMINFO\_H

#define MEM\_LINE\_MEMALL 1

#define MEM\_LINE\_MEMFREE 2

#define MEM\_LINE\_BUFFER 3

#define MEM\_LINE\_CACHED 4

#define MEM\_LINE\_SWAPCACHED 5

#define MEM\_LINE\_SWAPALL 14

#define MEM\_LINE\_SWAPFREE 15

void CreatePageTwo(GtkWidget \*notebook);

void DrawingMM();

void GetMmInfo();

int GetValueFromString(char \*string);

void DrawingSWAP();

typedef struct \_mem{

char MemTotal[20];

char MemFree[20];

char Buffers[20];

char Cached[20];

char SwapCached[20];

char SwapTotal[20];

char SwapFree[20];

}mem\_info;

GtkWidget \*draw\_mm;

GtkWidget \*window;

GtkWidget \*labelofmm;

GtkWidget \*mminfolabel;

GdkPixmap \*mm\_graph;

mem\_info memft;

static gint mmPoints[100];

static gint swapPoints[100];

static gfloat mm\_rate = 0.0;

static gfloat swap\_rate = 0.0;

gdouble memUseRatio, swapUseRatio;

char mmBuffer[2000];

GtkWidget \*window\_swap;

GtkWidget \*draw\_swap;

GdkPixmap \*swap\_graph;

char swapBuffer[200];

static gboolean mm\_expose\_event(GtkWidget \*widget, GdkEventExpose \*event, gpointer data)

{

gdk\_draw\_drawable(widget->window, widget->style->fg\_gc[GTK\_WIDGET\_STATE(widget)], mm\_graph, event->area.x, event->area.y, event->area.x, event->area.y, event->area.width, event->area.height);

return FALSE;

}

static gboolean mm\_configure\_event(GtkWidget \*widget, GdkEventConfigure \*event, gpointer data)

{

if (mm\_graph)

{

g\_object\_unref(mm\_graph);

}

mm\_graph = gdk\_pixmap\_new(widget->window, widget->allocation.width, widget->allocation.height, -1);

gdk\_draw\_rectangle(mm\_graph, widget->style->white\_gc, TRUE, 0, 0, widget->allocation.width, widget->allocation.height);

return TRUE;

}

static gboolean swap\_expose\_event(GtkWidget \*widget, GdkEventExpose \*event, gpointer data)

{

gdk\_draw\_drawable(widget->window, widget->style->fg\_gc[GTK\_WIDGET\_STATE(widget)], swap\_graph, event->area.x, event->area.y, event->area.x, event->area.y, event->area.width, event->area.height);

return FALSE;

}

static gboolean swap\_configure\_event(GtkWidget \*widget, GdkEventConfigure \*event, gpointer data)

{

if (swap\_graph)

{

g\_object\_unref(swap\_graph);

}

swap\_graph = gdk\_pixmap\_new(widget->window, widget->allocation.width, widget->allocation.height, -1);

gdk\_draw\_rectangle(swap\_graph, widget->style->white\_gc, TRUE, 0, 0, widget->allocation.width, widget->allocation.height);

return TRUE;

}

void CreatePageTwo(GtkWidget \*notebook)

{

GtkWidget \*vbox, \*frame, \*MMframe, \*SWAPframe;

frame = gtk\_frame\_new("");

gtk\_container\_set\_border\_width(GTK\_CONTAINER(frame), 10);

gtk\_widget\_show(frame);

gtk\_notebook\_append\_page(GTK\_NOTEBOOK(notebook), frame, gtk\_label\_new("Memory information"));

GtkWidget \*table = gtk\_table\_new(10, 10, TRUE);

gtk\_container\_add(GTK\_CONTAINER(frame), table);

MMframe = gtk\_frame\_new("MM使用曲线");

gtk\_table\_attach\_defaults(GTK\_TABLE(table), MMframe, 1, 9, 0, 5);

gtk\_widget\_show(MMframe);

SWAPframe = gtk\_frame\_new("swap 使用曲线");

gtk\_table\_attach\_defaults(GTK\_TABLE(table), SWAPframe, 1, 9, 5, 10);

gtk\_widget\_show(SWAPframe);

GetCpuInfo();

GetMmInfo();

gtk\_widget\_show(table);

draw\_mm = gtk\_drawing\_area\_new();

gtk\_widget\_set\_app\_paintable(draw\_mm, TRUE);

gtk\_drawing\_area\_size(GTK\_DRAWING\_AREA(draw\_mm), 40, 80);

gtk\_container\_add(GTK\_CONTAINER(MMframe), draw\_mm);

gtk\_widget\_show(draw\_mm);

DrawingMM();

draw\_swap = gtk\_drawing\_area\_new();

gtk\_widget\_set\_app\_paintable(draw\_swap, TRUE);

gtk\_drawing\_area\_size(GTK\_DRAWING\_AREA(draw\_swap), 40, 80);

gtk\_container\_add(GTK\_CONTAINER(SWAPframe), draw\_swap);

gtk\_widget\_show(draw\_swap);

DrawingSWAP();

}

void DrawingMM()

{

int i;

for (i = 0; i < 100; i++)

{

/\* 要取[a,b)之间的随机整数（包括a，但不包括b)，使用：

\* (rand() % (b - a)) + a \*/

mmPoints[i] = 999;

}

g\_signal\_connect(draw\_mm, "expose\_event",G\_CALLBACK(mm\_expose\_event), NULL);

g\_signal\_connect(draw\_mm, "configure\_event", G\_CALLBACK(mm\_configure\_event), NULL);

//gtk\_widget\_show(draw\_mm);

}

void DrawingSWAP()

{

int i;

for (i = 0; i < 100; i++)

{

/\* 要取[a,b)之间的随机整数（包括a，但不包括b)，使用：

\* (rand() % (b - a)) + a \*/

swapPoints[i] = 999;

}

g\_signal\_connect(draw\_swap, "expose\_event",G\_CALLBACK(swap\_expose\_event), NULL);

g\_signal\_connect(draw\_swap, "configure\_event", G\_CALLBACK(swap\_configure\_event), NULL);

}

gint refresh\_mm()

{

int i=0;

GdkGC \*gc\_chart\_mm = gdk\_gc\_new(draw\_mm->window);

GdkColor color;

color.red = 0xFFFF;

color.green = 0xFFFF;

color.blue = 0xFFFF;

gdk\_gc\_set\_rgb\_fg\_color(gc\_chart\_mm, &color);

int width1, height1, curPoint1, step1;

mm\_rate = memUseRatio;

gdk\_draw\_rectangle(mm\_graph, gc\_chart\_mm, TRUE, 0, 0,draw\_mm->allocation.width, draw\_mm->allocation.height);

width1 = draw\_mm->allocation.width;

height1 = draw\_mm->allocation.height;

curPoint1 = (int) (mm\_rate \* (double) height1);

mmPoints[99] = height1 - curPoint1;

int j;

for (j = 0; j < 99; j++)

mmPoints[j] = mmPoints[j + 1];

step1 = width1 / 99;

GdkGC \*gc = gdk\_gc\_new(GDK\_DRAWABLE(mm\_graph));

gdk\_color\_parse("red",&color);

gdk\_gc\_set\_foreground(gc, &color);

gdk\_gc\_set\_line\_attributes(gc, 1, GDK\_LINE\_SOLID, GDK\_CAP\_ROUND,GDK\_JOIN\_MITER);

for (j = 99; j >= 1; j--)

{

gdk\_draw\_line(GDK\_DRAWABLE(mm\_graph), gc, j \* step1, mmPoints[j], /\* 第一个点坐标 \*/

(j - 1) \* step1, mmPoints[j - 1]); /\* 第二个点坐标 \*/

}

gtk\_widget\_queue\_draw(draw\_mm);

GetMmInfo();

return TRUE;

}

gint refresh\_swap()

{

GdkGC \*gc\_chart\_swap = gdk\_gc\_new(draw\_swap->window);

GdkColor color;

color.red = 0xFFFF;

color.green = 0xFFFF;

color.blue = 0xFFFF;

gdk\_gc\_set\_rgb\_fg\_color(gc\_chart\_swap, &color);

int width1, height1, curPoint1, step1;

swap\_rate = swapUseRatio;

gdk\_draw\_rectangle(swap\_graph, gc\_chart\_swap, TRUE, 0, 0,draw\_swap->allocation.width, draw\_swap->allocation.height);

width1 = draw\_swap->allocation.width;

height1 = draw\_swap->allocation.height;

curPoint1 = (int) (swap\_rate \* (double) height1);

swapPoints[99] = height1 - curPoint1;

int j;

for (j = 0; j < 99; j++)

swapPoints[j] = swapPoints[j + 1];

step1 = width1 / 99;

GdkGC \*gc = gdk\_gc\_new(GDK\_DRAWABLE(swap\_graph));

gdk\_color\_parse("red",&color);

gdk\_gc\_set\_foreground(gc, &color);

gdk\_gc\_set\_line\_attributes(gc, 1, GDK\_LINE\_SOLID, GDK\_CAP\_ROUND,GDK\_JOIN\_MITER);

for (j = 99; j >= 1; j--)

{

gdk\_draw\_line(GDK\_DRAWABLE(swap\_graph), gc, j \* step1, swapPoints[j], /\* 第一个点坐标 \*/

(j - 1) \* step1, swapPoints[j - 1]); /\* 第二个点坐标 \*/

}

gtk\_widget\_queue\_draw(draw\_swap);

return TRUE;

}

//数据的获取

void GetMmInfo()

{

GetInfo("/proc/meminfo", "MemTotal", memft.MemTotal);

GetInfo("/proc/meminfo", "MemFree", memft.MemFree);

GetInfo("/proc/meminfo", "Buffers", memft.Buffers);

GetInfo("/proc/meminfo", "Cached", memft.Cached);

GetInfo("/proc/meminfo", "SwapCached", memft.SwapCached);

GetInfo("/proc/meminfo", "SwapTotal", memft.SwapTotal);

GetInfo("/proc/meminfo", "SwapFree", memft.SwapFree);

int mem\_total, mem\_free;

mem\_total = GetValueFromString(memft.MemTotal);

mem\_free = GetValueFromString(memft.MemFree);

memUseRatio = (double)(mem\_total - mem\_free) / mem\_total;

int swap\_total, swap\_free;

swap\_total = GetValueFromString(memft.SwapTotal);

swap\_free = GetValueFromString(memft.SwapFree);

swapUseRatio = (double)(swap\_total - swap\_free) / swap\_total;

return;

}

int GetValueFromString(char \*string)

{

char \*p = string;

int val = 0;

while(\*p >= '0' && \*p <= '9')

{

val = 10 \* val + \*p - '0';

p++;

}

return val;

}

#endif

Processinfo.h代码如下：

#ifndef \_\_PROCESSINFO\_H

#define \_\_PROCESSINFO\_H

GtkWidget \*clist;

#define PROC\_LINE\_NAME 1

#define PROC\_LINE\_PID 5

#define PROC\_LINE\_STATE 2

#define PROC\_LINE\_VMSIZE 13

//#define PROC\_LINE\_PRIORITY

typedef struct \_proc\_info\_st{

char name[256];

char pid[20];

char state[20];

char vmsize[20];

char priority[20];

char ppid[20];

}proc\_info;

void CreatePageThree(GtkWidget \*notebook);

gint ShowProcInfo(gpointer data);

void ReadProc(proc\_info \*ft, const char\* path);

void ReadLine(FILE \*fp, char \*buff, int bufflen, int line);

void SearchProcByPID(gpointer data);

void ProcessRun(gpointer data);

void SetEntryInfo(GtkWidget \*clist, gint row, gint column, GdkEventButton \*event, gpointer data);

static GtkWidget \*entry;

void CreatePageThree(GtkWidget \*notebook)

{

GtkWidget \*vbox, \*scroll, \*table, \*searchbutton1, \*searchbutton2, \*commondbutton, \*button;

table = gtk\_table\_new(10, 10, TRUE);

gtk\_notebook\_append\_page(GTK\_NOTEBOOK(notebook), table, gtk\_label\_new("Process information"));

gtk\_widget\_show(table);

gchar \*colname[6] = {"Processname", "PID", "PPID", "state", "Mem(KB)", "Priority"};

scroll = gtk\_scrolled\_window\_new(NULL, NULL);

gtk\_scrolled\_window\_set\_policy(GTK\_SCROLLED\_WINDOW(scroll), GTK\_POLICY\_AUTOMATIC, GTK\_POLICY\_ALWAYS);

//gtk\_notebook\_append\_page(GTK\_NOTEBOOK(notebook), scroll, gtk\_label\_new("Page3-process"));

gtk\_table\_attach\_defaults(GTK\_TABLE(table), scroll, 0, 10, 0, 9);

clist = gtk\_clist\_new\_with\_titles(6, colname);

g\_signal\_connect(GTK\_OBJECT(clist), "select\_row", GTK\_SIGNAL\_FUNC(SetEntryInfo), NULL);

//gtk\_signal\_connext(GTK\_OBJECT(clist), "select\_row", GTK\_SIGNAL\_FUNC(), NULL);

gtk\_clist\_set\_column\_width(GTK\_CLIST(clist), 0, 200);

gtk\_clist\_set\_column\_width(GTK\_CLIST(clist), 1, 100);

gtk\_clist\_set\_column\_width(GTK\_CLIST(clist), 2, 100);

gtk\_clist\_set\_column\_width(GTK\_CLIST(clist), 3, 100);

gtk\_clist\_set\_column\_width(GTK\_CLIST(clist), 4, 100);

gtk\_clist\_set\_column\_width(GTK\_CLIST(clist), 4, 110);

gtk\_container\_add(GTK\_CONTAINER(scroll), clist);

gtk\_widget\_show(clist);

entry = gtk\_entry\_new();

searchbutton1 = gtk\_button\_new\_with\_label("search by PID");

gtk\_container\_set\_border\_width(GTK\_CONTAINER(searchbutton1), 9);

searchbutton2 = gtk\_button\_new\_with\_label("run");

gtk\_container\_set\_border\_width(GTK\_CONTAINER(searchbutton2), 9);

button = gtk\_button\_new\_with\_label("refresh");

gtk\_container\_set\_border\_width(GTK\_CONTAINER(button), 5);

gtk\_table\_attach\_defaults(GTK\_TABLE(table), entry, 0, 4, 9, 10);

gtk\_table\_attach\_defaults(GTK\_TABLE(table), searchbutton1, 4, 6, 9, 10);

gtk\_table\_attach\_defaults(GTK\_TABLE(table), searchbutton2, 6, 8, 9, 10);

gtk\_table\_attach\_defaults(GTK\_TABLE(table), button, 8, 10, 9, 10);

gtk\_widget\_show(entry);

gtk\_widget\_show(searchbutton1);

gtk\_widget\_show(searchbutton2);

gtk\_widget\_show(button);

g\_signal\_connect(G\_OBJECT(searchbutton1), "clicked", G\_CALLBACK(SearchProcByPID), NULL);

g\_signal\_connect(G\_OBJECT(searchbutton2), "clicked", G\_CALLBACK(ProcessRun), NULL);

g\_signal\_connect(G\_OBJECT(button), "clicked", G\_CALLBACK(ShowProcInfo), NULL);

gtk\_timeout\_add(30000, ShowProcInfo, NULL);

//commondbutton

ShowProcInfo(NULL);

}

gint ShowProcInfo(gpointer data)

{

DIR \*dir;

struct dirent \*ptr;

dir = opendir("/proc");

gtk\_clist\_clear(GTK\_CLIST(clist));

while ((ptr = readdir(dir)) != NULL)

{

if ((ptr->d\_name[0] >= '0') && (ptr->d\_name[0] <= '9'))

{

proc\_info ft;

ReadProc(&ft, ptr->d\_name);

//printf("%s\n", ft.pid);

//printf("%s\n%s\n%s\n%s\n\n", ft.name, ft.pid, ft.state, ft.vmsize);

}

}

closedir(dir);

return TRUE;

}

void ReadProc(proc\_info \*ft, const char\* path)

{

FILE \*fp = NULL;

char file[512] = {0};

char buff[1024] = {0};

sprintf(file, "/proc/%s/status", path);

if(!(fp = fopen(file, "r")))

{

printf("read file %s wrong! exit\n", path);

return;

//exit(0); //MENTION :: CANNOT USE EXIT HERE, OR THIS PROGRAM WILL KINN ITSELF

}

char name[256];

ReadLine(fp, buff, 1024, PROC\_LINE\_NAME);

sscanf(buff, "%s %s", name, ft->name);

fseek(fp, 0, SEEK\_SET);

ReadLine(fp, buff, 1024, PROC\_LINE\_PID);

sscanf(buff, "%s %s", name, ft->pid);

fseek(fp, 0, SEEK\_SET);

ReadLine(fp, buff, 1024, PROC\_LINE\_STATE);

sscanf(buff, "%s %s", name, ft->state);

fseek(fp, 0, SEEK\_SET);

ReadLine(fp, buff, 1024, PROC\_LINE\_VMSIZE);

sscanf(buff, "%s %s", name, ft->vmsize);

if(strcmp(name, "VmSize:"))

{

ft->vmsize[0] = '0';

ft->vmsize[1] = '\0';

}

fclose(fp);

sprintf(file, "/proc/%s/stat", path); //Get priority

if(!(fp = fopen(file, "r")))

{

printf("read file %s wrong! exit\n", path);

return ;

//exit(0);

}

ReadLine(fp, buff, 1024, 1);

sscanf(buff, "%\*s %\*s %\*s %s %\*s %\*s %\*s %\*s %\*s %\*s %\*s %\*s %\*s %\*s %\*s %\*s %\*s %s", ft->ppid, ft->priority);

fclose(fp);

gchar \*list[1][6] = {

{ft->name, ft->pid, ft->ppid, ft->state, ft->vmsize, ft->priority}

};

gtk\_clist\_append((GtkCList\*) clist, list[0]);

gtk\_clist\_thaw((GtkCList \*) clist);

}

void ReadLine(FILE \*fp, char \*buff, int bufflen, int line)

{

char linebuff[bufflen];

int i;

for(i = 0; i < line - 1; i++)

{

fgets(linebuff, sizeof(linebuff), fp);

}

fgets(linebuff, sizeof(linebuff), fp);

memcpy(buff, linebuff, bufflen);

}

void SearchProcByPID(gpointer data)

{

const gchar \*pid;

pid = gtk\_entry\_get\_text(GTK\_ENTRY(entry));

puts(pid);

int i;

FILE \*fp = NULL;

char file[512] = {0};

char buf[9][256] = {0};

char msg[256];

sprintf(file, "/proc/%s/status", pid);

if(!(fp = fopen(file, "r")))

{

strcpy(msg, "pid is wrong\nplease input the right id of process!\n");

//puts(msg);

GtkWidget \*dialog ;

dialog = gtk\_message\_dialog\_new(NULL, GTK\_DIALOG\_DESTROY\_WITH\_PARENT, GTK\_MESSAGE\_INFO, GTK\_BUTTONS\_OK, msg, "");

gtk\_dialog\_run(GTK\_DIALOG(dialog));

gtk\_widget\_destroy(dialog);

return;

}

for(i = 0; i < 9; i++)

{

fgets(buf[i], sizeof(buf[i]), fp);

//puts(buf[i]);

}

strcpy(msg, buf[0]);

for(i = 1; i < 9; i++)

strcat(msg, buf[i]);

GtkWidget \*dialog;

dialog = gtk\_message\_dialog\_new(NULL, GTK\_DIALOG\_DESTROY\_WITH\_PARENT, GTK\_MESSAGE\_QUESTION, GTK\_BUTTONS\_YES\_NO, msg, "");

gtk\_window\_set\_title(GTK\_WINDOW(dialog), "杀死进程?");

GtkResponseType result = gtk\_dialog\_run(GTK\_DIALOG(dialog));

gtk\_widget\_destroy(dialog);

if (result == GTK\_RESPONSE\_YES || result == GTK\_RESPONSE\_APPLY)

{

//Kill the process

gchar cmd\_kill[50];

strcpy(cmd\_kill, "kill ");

strcat(cmd\_kill, pid);

system(cmd\_kill);

ShowProcInfo(NULL);

}

return;

}

void ProcessRun(gpointer data)

{

const gchar \*name;

name = gtk\_entry\_get\_text(GTK\_ENTRY(entry));

int id;

printf("%s", name);

if((id = fork()) == 0)

{

printf("hehe");

system(name);

return;

}

return;

}

void SetEntryInfo(GtkWidget \*clist, gint row, gint column, GdkEventButton \*event, gpointer data)

{

gchar \*pid;

gtk\_clist\_get\_text(GTK\_CLIST(clist), row, 1, &pid);

gtk\_entry\_set\_text(GTK\_ENTRY(entry), pid);

}

#endif

Sysinfo.h代码：

#include <unistd.h>

#include <string.h>

#include <errno.h>

#include <sys/utsname.h>

#ifndef \_\_SYSINFO\_H

#define \_\_SYSINFO\_H

void CreatePageFive(GtkWidget \*notebook); //Info

void GetSysTime(char \*runtime);

gint UpdateSystime(gpointer data);

//get the information of system

void CreatePageFive(GtkWidget \*notebook)

{

int s;

struct utsname u\_name;

s = uname(&u\_name);

char starttime[100];F

char sysinfo[1000];

if (s < 0)

{

printf("uname()failed\n");

exit(1);

}

/\*--------------------------编写系统信息--------------------------\*/

GetSysTime(starttime);

strcpy(sysinfo, "sysname: ");

strcat(sysinfo, u\_name.sysname);

strcat(sysinfo, "\nnodename: ");

strcat(sysinfo, u\_name.nodename);

strcat(sysinfo, "\nrelease: ");

strcat(sysinfo, u\_name.release);

strcat(sysinfo, "\nversion: ");

strcat(sysinfo, u\_name.version);

strcat(sysinfo, "\nmachine: ");

strcat(sysinfo, u\_name.machine);

strcat(sysinfo, "\n\n\nTotalMemory: ");

strcat(sysinfo, memft.MemTotal);

strcat(sysinfo, "FreeMemory: ");

strcat(sysinfo, memft.MemFree);

strcat(sysinfo, "Buffers: ");

strcat(sysinfo, memft.Buffers);

strcat(sysinfo, "SwapTotal: ");

strcat(sysinfo, memft.SwapTotal);

strcat(sysinfo, "SwapFree: ");

strcat(sysinfo, memft.SwapFree);

/\*-------------------------图形界面的编排---------------------------\*/

GtkWidget \*label, \*frame, \*vbox, \*timelabel;

frame = gtk\_frame\_new("");

gtk\_container\_set\_border\_width(GTK\_CONTAINER(frame), 10);

gtk\_widget\_show(frame);

gtk\_notebook\_append\_page(GTK\_NOTEBOOK(notebook), frame, gtk\_label\_new("System information"));

vbox = gtk\_vbox\_new(FALSE, 10);

gtk\_container\_set\_border\_width(GTK\_CONTAINER(vbox), 5);

label = gtk\_label\_new(sysinfo);

gtk\_container\_add(GTK\_CONTAINER(frame), vbox);

gtk\_widget\_show(label);

gtk\_box\_pack\_start(GTK\_BOX(vbox), label, TRUE, TRUE, 0);

timelabel = gtk\_label\_new(starttime);

gtk\_box\_pack\_start(GTK\_BOX(vbox), timelabel, TRUE, TRUE, 0);

gtk\_widget\_show(timelabel);

gtk\_timeout\_add(1000, UpdateSystime, timelabel);

}

void GetSysTime(char \*starttime)

{

char buf[50], \*ch;

time\_t nowtime;

struct tm \*p;

int i\_time, i\_runtime,i\_stime;

time(&nowtime);

p = localtime(&nowtime);

i\_time = p->tm\_hour\*3600 + p->tm\_min\*60+p->tm\_sec;

int fd = open("/proc/uptime", O\_RDONLY);

read(fd, buf, sizeof(buf));

close(fd);

ch = strtok(buf, " ");

i\_runtime = atoi(ch);

i\_stime = i\_time - i\_runtime;

int hour, min, sec;

char temp[2];

hour = i\_stime / 3600;

min = (i\_stime - 3600 \* hour) / 60;

sec = i\_stime - 3600\*hour - 60\*min;

strcpy(starttime, "System start from : ");

sprintf(temp, "%02d", hour);

strcat(starttime, temp);

strcat(starttime, ":");

sprintf(temp, "%02d", min);

strcat(starttime, temp);

strcat(starttime, ":");

sprintf(temp, "%02d", sec);

strcat(starttime, temp);

strcat(starttime, "\n");

hour = i\_runtime / 3600;

min = (i\_runtime - 3600 \* hour) / 60;

sec = i\_runtime - 3600\*hour - 60\*min;

strcat(starttime, "System runing for : ");

sprintf(temp, "%02d", hour);

strcat(starttime, temp);

strcat(starttime, ":");

sprintf(temp, "%02d", min);

strcat(starttime, temp);

strcat(starttime, ":");

sprintf(temp, "%02d", sec);

strcat(starttime, temp);

strcat(starttime, "\n\0");

}

gint UpdateSystime(gpointer data)

{

char systime[100];

GetSysTime(systime);

gtk\_label\_set\_text(GTK\_LABEL(data), systime);

return TRUE;

}

#endif

Makefile文件：

CC = gcc

all:

$(CC) -o a.out main.c `pkg-config --cflags --libs gtk+-2.0`

clean:

rm a.out

# 实验五 文件系统

## 实验目的

* 设计并实现一个模拟的文件系统（选做）

## 实验内容

* 基于一个大文件(如100M)，模拟磁盘
* 格式化，建立文件系统管理数据结构
* 实现文件/目录创建/删除，目录显示等基本功能(可自行扩充文件读/写、用户登录、权限控制、读写保护等其他功能)

## 实验设计

我的模拟的文件系统主要是仿照Linux现有的功能实现的，虽然之前还思考过身份认证的功能，但由于时间和能力的局限所以并未完成。

在完成的功能，主要包括文件创建、文件查看、文件跳转、文件夹创建、文件打开、文件关闭、文件读写操作以及文件复制删除功能。

## 开发环境

* 主机系统版本：Window 10 64位
* 主机内存：8G
* Intel(R) Core(TM) i5-4300M CPU @2.60GHz 2.60 GHz
* 虚拟机：vmware workstation
* 虚拟机操作系统 Ubuntu16.04 64位
* 虚拟机内存：4G
* 虚拟机硬盘大小：80G

## 实验设计

## 实验调试

## 实验调试及心得

这是实验时候做的功能界面，其中功能的调用是输入右边的字符串，而不是数字，数字只是仅供功能数量的参考，如图5.1。

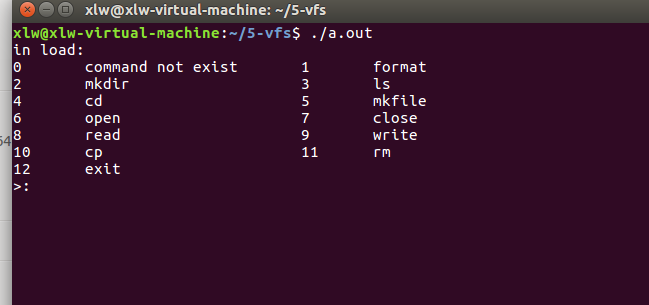


图5.1

#### 功能1：模拟磁盘格式化

这个没法显示，单单是对磁盘格式化。

#### 功能2：创建文件目录

文件目录是指：为实现“按名存取”，必须建立文件名与辅存空间中物理地址的对应关系，体现这种对应关系的数据结构称为文件目录。

功能2创建文件夹，即文件目录。创建成的文件夹可由功能3查看文件目录，我先创建一个名为222的文件夹，如图5.2所示。

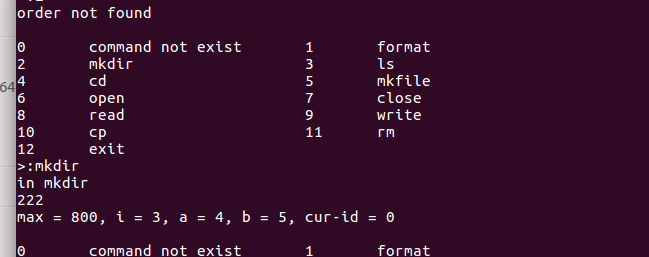


图5.2

#### 功能3：查看文件目录

查看该目录下的文件，显然在功能2我创建了一个名为222的文件夹，现在可以在其中看到，如图5.3。

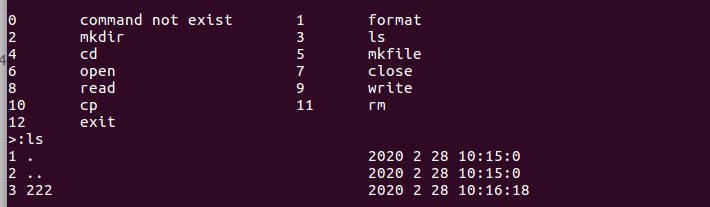


图5.3

#### 功能4：跳转目录

文件跳转命令cd，在图5.3中有个222的文件夹，因此输入222，如图5.4。接着再ls，发现已经跳转成功，如图5.5。。

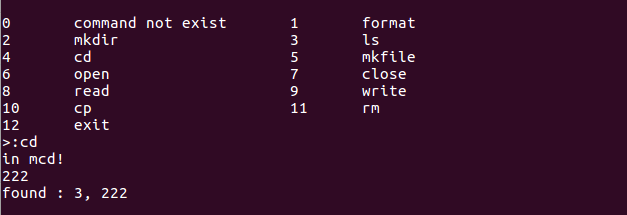


图5.4

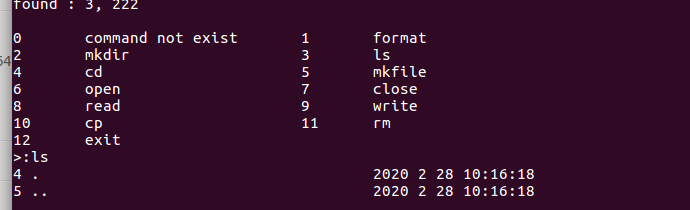


图5.5

#### 功能5：文件创建

Mkfile指令可以创建一个文件，在222文件夹当中，创建一个名为123的文件，可以在ls当中看到，123文件成功的创建，如图5.7。

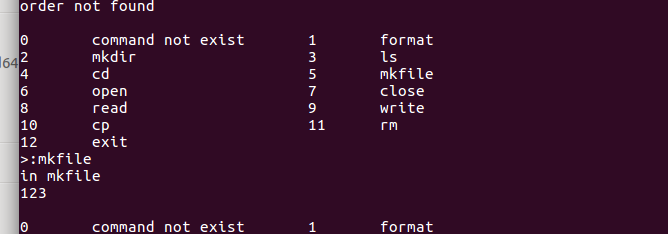


图5.6

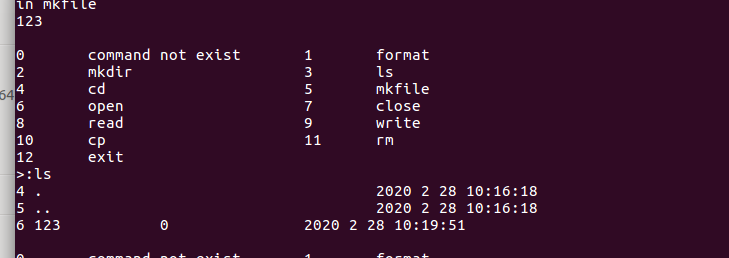


图5.7

#### 功能6：文件打开

文件打开功能open打开之前的123文件，如图5.8。

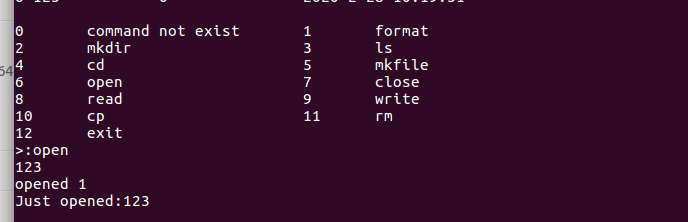


图5.8

#### 功能7：文件关闭

打开后关闭，如图5.9。

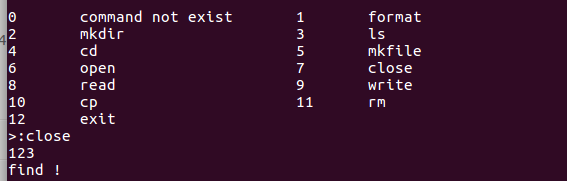


图5.9

#### 功能8：读操作

由于是个空文件，所以读出来也是0 chars，如图5.10。

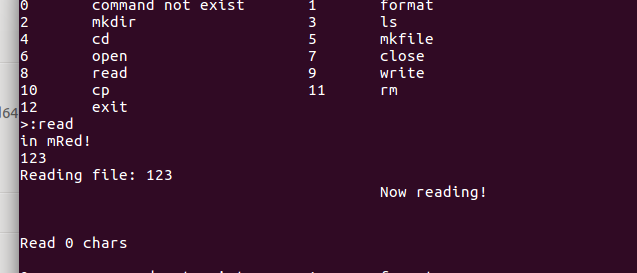


图5.10

#### 功能9：写操作

想文件当中写入信息，且结尾用#表示终止，如图5.11。写入之后再次读取，可以看到之前的写入功能成功实现，如图5.12。

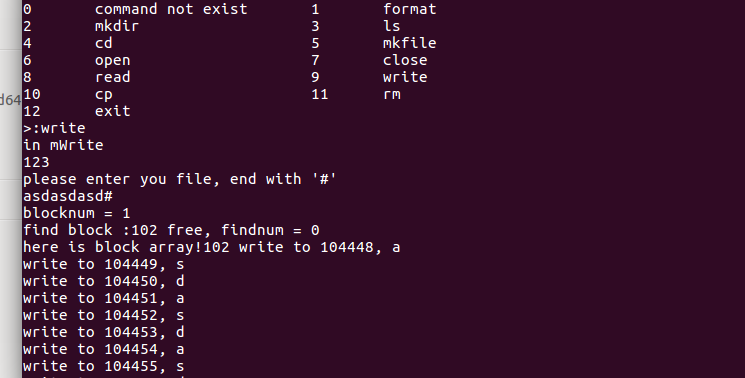


图5.11

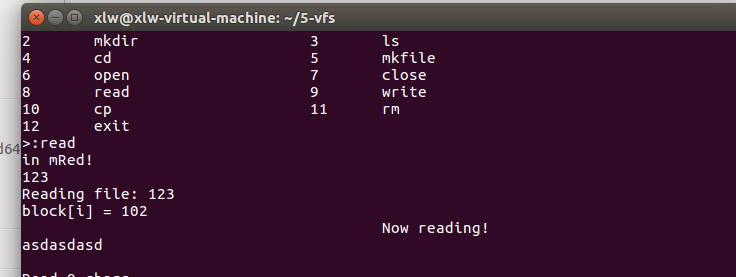


图5.12

#### 功能10：文件复制

打开一个文件进行复制，若不存在则报错，然后选择复制到的文件，如果不存在则创建一个。我们再去检测文件是否复制成功，如图5.13。

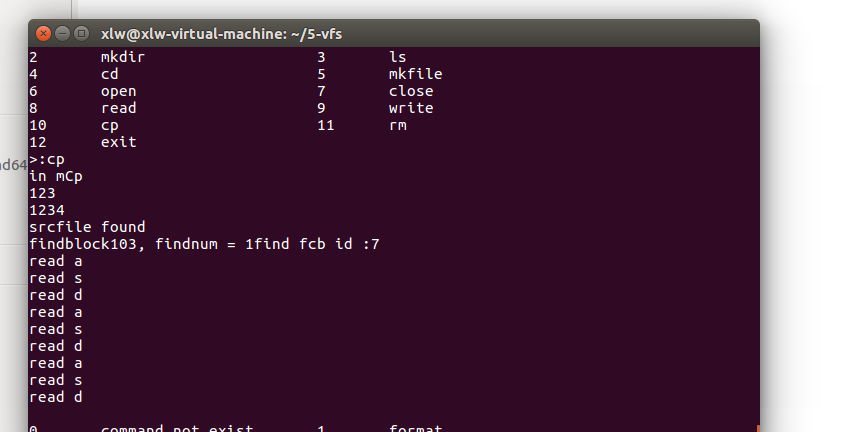


图5.13

#### 功能11：文件删除

把之前的123文件删除，如图5.14，然后再ls查看，发现文件已经成功的删除，如图5.15。



图5.14

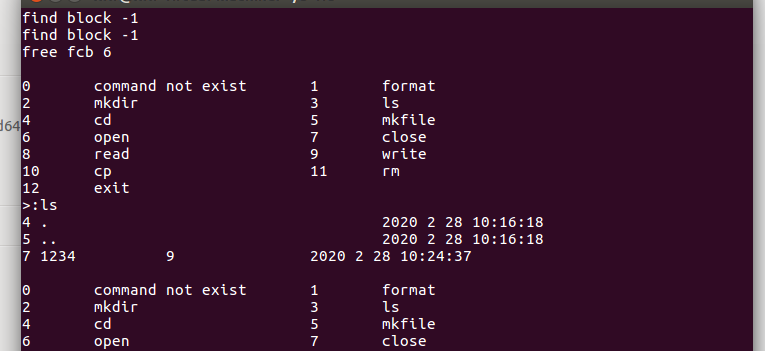


图5.15

#### 功能12：exit

退出文件系统，嗯，也没什么特殊的，如图5.16。

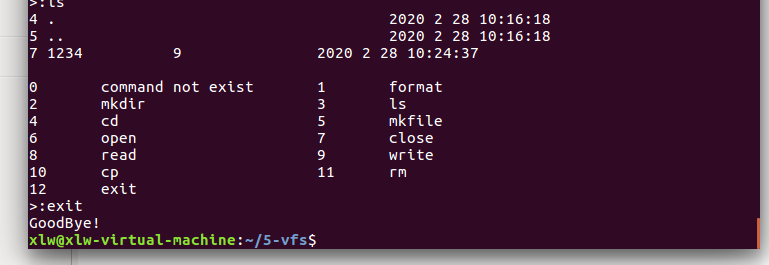


图5.16

## 附录 实验代码

mRead.h代码：

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*--------------------读文件-----------------------\*/

int mRead()

{

printf("in mRed!\n");

int i, j;

FCB \*fcb = (FCB \*)(fdisk + BLOCKSIZE + BLOCKSIZE);

char \*flag = (char \*)(fdisk + BLOCKSIZE);

FCB \*cur = cur\_dir;

FCB \*temp;

char filename[20];

scanf("%s", filename);

for(i = 0; i < 10; i++)

{

if(cur->record[i] != -1)

{

temp = fcb + cur->record[i];

if(strcmp(temp->filename, filename) == 0)

{

break;

}

}

}

if(temp->isdir == 1 || i == 10)

return ERROR\_READ\_FILE\_NOT\_FOUND;

printf("Reading file: %s\n", temp->filename);

int block[10], blocknum = 0, sizeread = 0, blockread = 0, readingblock;

//temp is the file to read

blocknum = temp->length;

//record the file's block-id

for(i = 0; i < blocknum; i++)

{

block[i] = temp->record[i];

printf("block[i] = %d\n", block[i]);

}

char \*reading;

printf("\t\t\t\t\tNow reading!\n");

while(sizeread < temp->size)

{

readingblock = sizeread / BLOCKSIZE;

reading = fdisk + 1024 \* block[readingblock] + sizeread % BLOCKSIZE; //array to the char that is reading

printf("%c", \*reading);

sizeread++;

}

printf("\n\nRead %d chars\n", sizeread);

return 1;

}

mRm.h文件：

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*--------------------读文件-----------------------\*/

int mRead()

{

printf("in mRed!\n");

int i, j;

FCB \*fcb = (FCB \*)(fdisk + BLOCKSIZE + BLOCKSIZE);

char \*flag = (char \*)(fdisk + BLOCKSIZE);

FCB \*cur = cur\_dir;

FCB \*temp;

char filename[20];

scanf("%s", filename);

for(i = 0; i < 10; i++)

{

if(cur->record[i] != -1)

{

temp = fcb + cur->record[i];

if(strcmp(temp->filename, filename) == 0)

{

break;

}

}

}

if(temp->isdir == 1 || i == 10)

return ERROR\_READ\_FILE\_NOT\_FOUND;

printf("Reading file: %s\n", temp->filename);

int block[10], blocknum = 0, sizeread = 0, blockread = 0, readingblock;

//temp is the file to read

blocknum = temp->length;

//record the file's block-id

for(i = 0; i < blocknum; i++)

{

block[i] = temp->record[i];

printf("block[i] = %d\n", block[i]);

}

char \*reading;

printf("\t\t\t\t\tNow reading!\n");

while(sizeread < temp->size)

{

readingblock = sizeread / BLOCKSIZE;

reading = fdisk + 1024 \* block[readingblock] + sizeread % BLOCKSIZE; //array to the char that is reading

printf("%c", \*reading);

sizeread++;

}

printf("\n\nRead %d chars\n", sizeread);

return 1;

}

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*--------------------写文件-----------------------\*/

int mWrite()

{

fp = fopen("VirtualDisk.txt", "wb+");

printf("请输入写入文件\n");

int i, j;

char filename[20];

scanf("%s", filename);

FCB \*fcb = (FCB \*)(fdisk + BLOCKSIZE + BLOCKSIZE);

char \*flag = (char \*)(fdisk + BLOCKSIZE);

FCB \*cur = cur\_dir;

FCB \*temp;

for(i = 0; i < 10; i++)

{

if(cur->record[i] != -1)

{

temp = fcb + cur->record[i];

if(strcmp(temp->filename, filename) == 0)

{

break;

}

}

}

if(temp->isdir == 1 || i == 10)

return ERROR\_WRITE\_FILE\_NOT\_FOUND;

//foundfile: temp

printf("please enter you file, end with '#'\n");

char ch;

char blockcontent[10][BLOCKSIZE];

for(i = 0; i < 10; i++)

{

for(j = 0; j < BLOCKSIZE; j++)

blockcontent[i][j] = '\0';

}

int count = 0, blocknum = 0;

getchar();

while((ch = getchar()) != '#')

{

blockcontent[blocknum][count] = ch;

count++;

if(count >= BLOCKSIZE)

{

blocknum++;

count = count - BLOCKSIZE;

}

}

blocknum++;

int block[10] = {-1};

int findnum = 0;

//find count block's space that can write file

printf("blocknum = %d\n", blocknum);

//getchar();

for(i = 0; i < BLOCKSIZE; i++)

{

if(flag[i] == FREE)

{

flag[i] = USED;

printf("find block :%d free, findnum = %d\n", i, findnum);

block[findnum] = i;

findnum++;

if(findnum == blocknum)

break;

}

}

printf("here is block array!");

for(i = 0; i < blocknum;i++)

{

printf("%d ", block[i]);

}

int overflag = 0, writenum = 0;;

for(i = 0; i < blocknum; i++)

{

int k = block[i];

for(j = 0; j < BLOCKSIZE; j++)

{

printf("write to %d, %c\n", 1024 \* k + j, blockcontent[i][j]);

fdisk[1024 \* k+j] = blockcontent[i][j];

writenum++;

if(writenum == count)

{

overflag = 1;

break;

}

}

if(overflag == 1)

break;

}

timep = time(NULL);

printf("write id : %d, blocknum = %d\n", temp->id, blocknum);

temp->size = count;

temp->data\_time = \*localtime(&timep);

printf("count = %d, size = %d\n", count, temp->size);

for(i = 0; i < blocknum; i++)

{

temp->record[i] = block[i];

printf("to block %d\n", temp->record[i]);

}

for(i = blocknum; i < 10; i++)

temp->record[i] = -1;

temp->length = blocknum;

fwrite(fdisk, BLOCKSIZE, 1024, fp);

fclose(fp);

return 1;

}

#ifndef NODE\_H\_

#define NODE\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define BLOCKSIZE 1024

#define SIZE 1024 \* 1024 //1 M

#define END 65535 //EOF

#define FREE 48

#define USED 49

#define FCBBLOCKNUM 100

#define MAXLENGTH 10

#define MAXOPEN 5

#define COMMANDNUM 14

/\*-------------------------报错----------------------------\*/

#define ERROR\_DISK\_NO\_SPACE 99

#define ERROR\_MKDIR\_IS\_FULL 100

#define ERROR\_MKDIR\_CONFLICT\_NAME 101

#define ERROR\_MKDIR\_USELESS\_DIRNAME 102

#define ERROR\_MKFILE\_IS\_FULL 103

#define ERROR\_CD\_PATH\_NOT\_FOUND 104

#define ERROR\_RM\_FILE\_NOT\_FOUND 105

#define ERROR\_RM\_DIR\_IS\_NOT\_EMPTY 106

#define ERROR\_READ\_FILE\_NOT\_FOUND 107

#define ERROR\_READ\_FILE\_IS\_EMPTY 108

#define ERROR\_WRITE\_FILE\_NOT\_FOUND 109

#define ERROR\_OPEN\_FILE\_NOT\_FOUND 110

#define ERROR\_OPEN\_FILE\_IS\_OPENED 111

#define ERROR\_OPEN\_OPENTABLE\_IS\_FULL 112

#define ERROR\_CLOSE\_FILE\_NOT\_FOUND 113

#define ERROR\_CLOSE\_FILE\_NOT\_OPENED 114

#define ERROR\_CP\_SRCFILE\_NOT\_FOUND 115

#define ERROR\_CP\_DSTFILE\_IS\_EXIST 116

#define ERROR\_CP\_IS\_FULL 117

#define MAXFILENUM FCBBLOCKNUM \* BLOCKSIZE \* sizeof(char) / sizeof(FCB)

/\*-------------------------文件的属性----------------------------\*/

//名称-------1

//标识符-----2

//类型-------3

//位置-------4

//大小-------5

//保护-------6

//时间，日期和用户标识-------7

/\*-------------------文件控制块(FCB)--------------------\*/

typedef struct \_FCB{

char filename[8]; //名称

int id; //标识符，标识文件系统内文件的唯一标签

char isdir; //1:yes 2 no

unsigned int firstblock;

struct tm data\_time; //时间，日期和用户标识。在time.h中定义

unsigned int size; //大小

int record[10];

int length;

}FCB;

typedef struct \_OPENTABLE{

char name[20];

struct tm data\_time;

int size;

}OPENTABLE;

//记载文件打开数量

typedef struct \_FILEOPEN{

OPENTABLE opened[MAXOPEN];

int opennum;

}FILEOPEN;

/\*-------------------------全局变量----------------------------\*/

FCB \*root; //根FCB

FCB \*cur\_dir; //当前的FCB

FILEOPEN fileopened; //记载打开文件数量

FILE \*fp;

int id = 1; //序号

int fd = -1;

char currentdir[80];

int currentid = 0;

char \*fdisk;

time\_t timep;

/\*-------------------------函数声明----------------------------\*/

int GetCommandType(char \*command);

void LoadSys();

void InitSystem();

void mFormat();

int mMkdir();

void mLs(void);

int mCd(void);

int mMkfile(void);

int mOpen(void);

int mClose(void);

int mRead(void);

int mWrite(void);

int mCp(void);

int mRm(void);

void mExit(void);

void ErrorConfig(int errorcode);

void FreeFCB(FCB \*fcb);

int BackUp(void);

#endif

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

int BackUp(void)

{

char s[50] = "cp VirtualDisk.txt backup.disk";

system(s);

return 1;

}

#ifndef LOADSYS\_H\_

#define LOADSYS\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*------------------------报错------------------------\*/

void ErrorConfig(int errorcode)

{

switch(errorcode)

{

case ERROR\_MKDIR\_IS\_FULL:

printf("Error: the dir is full now, cannot create dir in it!\n");

break;

case ERROR\_MKDIR\_CONFLICT\_NAME:

printf("Error: dir exists\n");

break;

case ERROR\_MKDIR\_USELESS\_DIRNAME:

printf("Error: dir name is invalid\n");

break;

case ERROR\_CD\_PATH\_NOT\_FOUND:

printf("Error: please enter the right path!\n");

break;

case ERROR\_RM\_FILE\_NOT\_FOUND:

printf("Error: please enter the right filename!\n");

break;

case ERROR\_RM\_DIR\_IS\_NOT\_EMPTY:

printf("Error: cannot delete a directory with items!\n");

break;

case ERROR\_READ\_FILE\_NOT\_FOUND:

printf("Error: cannot read, file is not found!\n");

break;

case ERROR\_READ\_FILE\_IS\_EMPTY:

printf("Error: cannot read, file is empty!");

break;

case ERROR\_WRITE\_FILE\_NOT\_FOUND:

printf("Error: cannot write, file is not found!\n");

break;

case ERROR\_OPEN\_FILE\_NOT\_FOUND:

printf("Error: cannot open file, file not found!\n");

break;

case ERROR\_OPEN\_FILE\_IS\_OPENED:

printf("Error: cannot open file, file is opened!\n");

break;

case ERROR\_CLOSE\_FILE\_NOT\_OPENED:

printf("Error: cannot close file, file is not opened!\n");

break;

default:

return;

}

}

#ifndef INITSYSTEM\_H\_

#define INITSYSTEM\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*------------------------系统初始化------------------------\*/

void InitSystem()

{

int i;

//为fdisk分配空间

fdisk = (char \*)malloc(SIZE \* sizeof(char)); //SIZE=1024\*1024

printf("sizeof(fdisk) = %d\n", (int)sizeof(fdisk));

mFormat();

}

#ifndef LOADSYS\_H\_

#define LOADSYS\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*-------------------------加载----------------------------\*/

void LoadSys()

{

fileopened.opennum = 0;

if((fp = fopen("VirtualDisk.txt", "rb")) == NULL)

{

printf("LoadSys : file doesn't exist!\n");

InitSystem();

return ;

}

printf("in load:");

fdisk = (char \*)malloc(SIZE \* sizeof(char));

cur\_dir = (FCB \*)(fdisk + BLOCKSIZE + BLOCKSIZE);

root = cur\_dir;

fread(fdisk, BLOCKSIZE, 1024, fp);

fclose(fp);

if(fdisk[0] != '$')

{

printf("%c doesn't find the disk!\n", fdisk[0]);

mFormat();

}

}

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*------------------------------跳转目录----------------------\*/

int mCd()

{

printf("请输入跳转的地址!\n");

int i;

char dir[20];

scanf("%s", dir);

FCB \*fcb = (FCB \*)(fdisk + BLOCKSIZE + BLOCKSIZE);

char \*flag = (char \*)(fdisk + BLOCKSIZE);

FCB \*cur = cur\_dir;

FCB \*temp;

if(strcmp(dir, "\\") == 0)

{

cur\_dir = root;

return 1;

}

if(strcmp(dir, ".") == 0)

return 1;

else if(strcmp(dir, "..") == 0)

{

printf("found %d\t", cur->record[i]);

temp = fcb + cur->record[1];

cur\_dir = fcb + temp->record[0];

printf("found : %d", temp->record[0]);

return 1;

}

for(i = 2; i < 10; i++)

{

temp = fcb + cur->record[i];

if(strcmp(temp->filename, dir) == 0)

{

printf("found : %d, %s\n", temp->id,temp->filename);

break;

}

}

if(i == 10 || temp->isdir == 0)

return ERROR\_CD\_PATH\_NOT\_FOUND;

cur\_dir = temp;

return 1;

}

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*--------------------关闭文件-----------------------\*/

int mClose()

{

int i;

char filename[20];

scanf("%s", filename);

int opennum = fileopened.opennum;

for(i = 0; i < opennum; i++)

{

if(strcmp(fileopened.opened[i].name, filename) == 0)

break;

}

if(i == opennum)

return ERROR\_CLOSE\_FILE\_NOT\_OPENED;

printf("find !\n");

for( ; i < opennum - 1; i++)

{

fileopened.opened[i] = fileopened.opened[i + 1];

}

fileopened.opennum--;

//delete opened file from fileopentable

return 1;

}

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

int mCp()

{

printf("in mCp\n");

int i,j, fcbid;

char srcfile[20], dstfile[20];

FCB \*fcb = (FCB \*)(fdisk + BLOCKSIZE + BLOCKSIZE);

char \*flag = (char \*)(fdisk + BLOCKSIZE);

FCB \*cur = cur\_dir;

FCB \*tempsrc, \*tempdst;

scanf("%s %s", srcfile, dstfile);

for(i = 0; i < 10; i++)

{

if(cur->record[i] != -1)

{

tempsrc = fcb + cur->record[i];

if(strcmp(tempsrc->filename, srcfile) == 0)

{

break;

}

}

}

if(tempsrc->isdir == 1 || i == 10)

return ERROR\_CP\_SRCFILE\_NOT\_FOUND;

for(i = 0; i < 10; i++)

{

if(cur->record[i] != -1)

{

tempdst = fcb + cur->record[i];

if(strcmp(tempdst->filename, dstfile) == 0)

{

break;

}

}

}

if(i != 10)//didnot think about dir is full

return ERROR\_CP\_DSTFILE\_IS\_EXIST;

for(i = 0; i < MAXFILENUM; i++)

{

if(fcb[i].id == -1)

break;

}

if(i == MAXFILENUM)

return ERROR\_DISK\_NO\_SPACE;

fcbid = i;

tempdst = fcb + i;

printf("srcfile found\n");

int blocknum = tempsrc->length;

int findnum = 0;

int block[10] = {0};

//find block that is free and store the blocks to block[10]

if(blocknum == 0)

{

timep = time(NULL);

tempdst->id = fcbid;

strcpy(tempdst->filename, dstfile);

tempdst->isdir = 0;

tempdst->data\_time = \*localtime(&timep);

tempdst->firstblock = 0;

tempdst->size = tempsrc->size;

tempdst->length = tempsrc->length;

//change the info of dir

cur->record[cur->length] = fcbid;

cur->length = cur->length + 1;

cur->data\_time = \*localtime(&timep);

return 1;

}

for(i = 0; i < BLOCKSIZE; i++)

{

if(flag[i] == FREE)

{

flag[i] = USED;

printf("findblock%d, findnum = %d", i, findnum+1);

block[findnum] = i;

findnum++;

if(findnum == blocknum)

break;

}

}

printf("find fcb id :%d\n", fcbid);

//set fcb info

timep = time(NULL);

tempdst->id = fcbid;

strcpy(tempdst->filename, dstfile);

tempdst->isdir = 0;

tempdst->data\_time = \*localtime(&timep);

tempdst->firstblock = 0;

tempdst->size = tempsrc->size;

tempdst->length = tempsrc->length;

//change the info of dir

cur->record[cur->length] = fcbid;

cur->length = cur->length + 1;

cur->data\_time = \*localtime(&timep);

for(i = 0; i < tempdst->length; i++)

{

tempdst->record[i] = block[i];

}

//copying file

char \*reading;

char blockcontent[10][BLOCKSIZE];

int copycount = 0, readingblock;

int count = tempsrc->size;

//read

while(copycount < count)

{

readingblock = copycount / BLOCKSIZE;

reading = fdisk + 1024 \* tempsrc->record[readingblock] + copycount % BLOCKSIZE; //array to the char that is reading

printf("read %c\n", \*reading);

blockcontent[readingblock][copycount % BLOCKSIZE] = \*reading;

copycount++;

}

//write

int overflag = 0, writenum = 0;

for(i = 0; i < blocknum; i++)

{

int k = block[i];

for(j = 0; j < BLOCKSIZE; j++)

{

fdisk[1024 \* k + j] = blockcontent[i][j];

writenum++;

if(writenum == count)

{

overflag = 1;

break;

}

}

if(overflag = 1)

break;

}

fopen("VirtualDisk.txt", "wb+");

fwrite(fdisk, BLOCKSIZE, 1024, fp);

fclose(fp);

return 1;

}

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*-----------------------结束退出--------------------------\*/

void mExit()

{

printf("欢迎下次使用!\n");

exit(0);

}

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*------------------------磁盘格式化------------------------\*/

void mFormat()

{

fp = fopen("VirtualDisk.txt", "wb+");

int i;

char \*p = fdisk + BLOCKSIZE;

for(i = 0; i < SIZE; i++)

fdisk[i] = '\_';

fdisk[0] = '$';

for(i = 0; i < 2 + FCBBLOCKNUM; i++)

p[i] = USED;

for(; i < BLOCKSIZE; i++)

p[i] = FREE;

/\*for(i = 0; i < BLOCKSIZE; i++)

printf("%d", p[i]);\*/

printf("\n\n\n");

FCB \*fcb = (FCB \*)(fdisk + BLOCKSIZE + BLOCKSIZE);

cur\_dir = fcb;

root = cur\_dir;

timep = time(NULL);

for(i = 0; i < MAXFILENUM; i++)

{

fcb[i].id = -1;

strcpy(fcb[i].filename, "");

fcb[i].isdir = 0;

fcb[i].data\_time = \*localtime(&timep);

fcb[i].firstblock = 0;

fcb[i].size = 0;

fcb[i].length = 0;

int j;

for(j = 0; j < 10; j++)

fcb[i].record[j] = -1;

//printf("set %2d: %2d, %s\n", i, fcb[i].id, fcb[i].filename);

}

fcb[0].id = 0;

strcpy(fcb[0].filename, "\\"); //"home"

fcb[0].isdir = 1;

fcb[0].data\_time = \*localtime(&timep);

fcb[0].firstblock = 0;

fcb[0].size = 0;

fcb[0].length = 2;

fcb[0].record[0] = 1; //"."

fcb[0].record[1] = 2; //".."

fcb[1].id = 1;

strcpy(fcb[1].filename, ".");

fcb[1].isdir = 1;

fcb[1].data\_time = \*localtime(&timep);

fcb[1].firstblock = 0;

fcb[1].size = 0;

fcb[1].length = 1;

fcb[1].record[0] = 0; //"home"

fcb[2].id = 2;

strcpy(fcb[2].filename, "..");

fcb[2].isdir = 1;

fcb[2].data\_time = \*localtime(&timep);

fcb[2].firstblock = 0;

fcb[2].size = 0;

fcb[2].length = 1;

fcb[2].record[0] = 0; //"home"

printf("sizeof(char) = %d, sizeof(FCB) = %d, MAXFILENUM = %d\n", (int)sizeof(char), (int)sizeof(FCB), (int)(MAXFILENUM));

/\*for(i = 0; i < MAXFILENUM; i++)

{

printf("i = %2d, %d", i, fcb[i].id);

if(fcb[i].id != -1)

{

printf("%s\n", fcb[i].filename);

}

}

for(i = 0; i < SIZE; i++)

{

if(i % BLOCKSIZE == 0)

printf("\n\n\n\n");

printf("%c",fdisk[i]);

}\*/

fwrite(fdisk, BLOCKSIZE, 1024, fp);

fseek(fp, 0, SEEK\_SET);

return;

}

#ifndef MFORMAT\_H\_

#define MFORMAT\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

/\*------------------------------查询文件信息----------------------\*/

void mLs()

{

FCB \*fcb = (FCB \*)(fdisk + BLOCKSIZE + BLOCKSIZE);

FCB temp;

int i;

for(i = 0; i < 10; i++)

{

int pos = cur\_dir->record[i];

if(pos >= 0)

{

temp = fcb[pos];

if(temp.isdir)

{

printf("%d ", pos);

printf("%s\t\t\t\t\t", temp.filename);

printf ("%d %d %d ", (1900 + temp.data\_time.tm\_year), (1 + temp.data\_time.tm\_mon), temp.data\_time.tm\_mday);

printf("%d:%d:%d\n", temp.data\_time.tm\_hour, temp.data\_time.tm\_min, temp.data\_time.tm\_sec);

}

else

{

printf("%d ", pos);

printf("%s\t\t%d\t\t", temp.filename, temp.size);

printf ("%d %d %d ", (1900 + temp.data\_time.tm\_year), (1 + temp.data\_time.tm\_mon), temp.data\_time.tm\_mday);

printf("%d:%d:%d\n", temp.data\_time.tm\_hour, temp.data\_time.tm\_min, temp.data\_time.tm\_sec);

}

}

}

return;

}

#ifndef LOADSYS\_H\_

#define LOADSYS\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "Node.h"

#endif

int mMkdir()

{

fp = fopen("VirtualDisk.txt", "wb+");

printf("in mkdir\n");

int i, a, b, j;

char dirname[20];

scanf("%s", dirname);

FCB \*fcb = (FCB \*)(fdisk + BLOCKSIZE + BLOCKSIZE);

char \*flag = (char \*)(fdisk + BLOCKSIZE);

FCB \*cur = cur\_dir;

FCB \*temp;

if(cur->length == MAXLENGTH)

return ERROR\_MKDIR\_IS\_FULL;

//find if the name of dir is exist

for (i = 0; i < cur->length; i++)

{

temp = fcb + cur->length;

if(strcmp(temp->filename, dirname) == 0)

return ERROR\_MKDIR\_CONFLICT\_NAME;

}

for(i = 0; i < MAXFILENUM; i++)

{

if(fcb[i].id == -1)

break;

}

timep = time(NULL);

fcb[i].id = i;

strcpy(fcb[i].filename, dirname);

fcb[i].isdir = 1;

fcb[i].data\_time = \*localtime(&timep);

fcb[i].firstblock = 0;

fcb[i].size = 0;

fcb[i].length = 2;

//todo :record[0] record[1];

//sub dir of dirname

for(a = i; a < MAXFILENUM; a++)

{

if(fcb[a].id == -1)

break;

}

for(j = 2; j < 10; j++)

fcb[i].record[j] = -1;

fcb[a].id = a;

strcpy(fcb[a].filename, ".");

fcb[a].isdir = 1;

fcb[a].data\_time = \*localtime(&timep);

fcb[a].firstblock = 0;

fcb[a].size = 0;

fcb[a].length = 1;

fcb[a].record[0] = i;

for(j = 1; j < 10; j++)

fcb[a].record[j] = -1;

for(b = a; b < MAXFILENUM; b++)

{

if(fcb[b].id == -1)

break;

}

fcb[b].id = b;

strcpy(fcb[b].filename, "..");

fcb[b].isdir = 1;

fcb[b].data\_time = \*localtime(&timep);

fcb[b].firstblock = 0;

fcb[b].size = 0;

fcb[b].length = 1;

fcb[b].record[0] = cur->id;

for(j = 1; j < 10; j++)

fcb[b].record[j] = -1;

fcb[i].record[0] = a;

fcb[i].record[1] = b;

printf("max = %d, i = %d, a = %d, b = %d, cur-id = %d\n",(int)(MAXFILENUM), i, a, b, cur->id);

cur->record[cur->length] = i;

cur->length = cur->length + 1;

cur->data\_time = \*localtime(&timep);

fwrite(fdisk, BLOCKSIZE, 1024, fp);

fclose(fp);

return 1;

}

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "BackUp.h"

#include "ErrorConfig.h"

#include "Node.h"

#include "LoadSys.h"

#include "InitSystem.h"

#include "mcd.h"

#include "mCp.h"

#include "mClose.h"

#include "mExit.h"

#include "mFormat.h"

#include "mLs.h"

#include "mMkdir.h"

#include "mMkfile.h"

#include "mOpen.h"

#include "mRead.h"

#include "mRm.h"

#include "mWrite.h"

/\*------------------------匹配命令----------------------\*/

//0 command not exist

//1 format

//2 mkdir

//3 ls

//4 cd

//5 mkfile

//6 open

//7 close

//8 read

//9 write

//10 cp

//11 rm

//12 exit

int GetCommandType(char \*command)

{

char cmd[COMMANDNUM][10] = {

"", "format", "mkdir", "ls", "cd",

"mkfile", "open", "close", "read",

"write", "cp", "rm", "exit", "backup"

};

int i;

for(i = 0; i < COMMANDNUM; i++)

{

if(strcmp(cmd[i], command) == 0)

return i;

}

return 0;

}

/\*---------------------------主程序------------------------\*/

void main(void)

{

int ordertype, error;

char command[10];

LoadSys();

while(1)

{

printf("\n1 format 2 mkdir\n");

printf("3 ls 4 cd\n");

printf("5 mkfile 6 open\n");

printf("7 close 8 read\n");

printf("9 write 10 cp\n");

printf("11 rm 12 exit\n");

printf("13 backup\n");

printf("请输入英文命令\n");

printf(">:");

scanf("%s", command);

ordertype = GetCommandType(command);

switch (ordertype)

{

case 1://format

mFormat();

break;

case 2://mkdir（创建子目录）

error = mMkdir();

ErrorConfig(error);

break;

case 3://ls（查看参数）

mLs();

break;

case 4://cd（切换命令）

error = mCd();

ErrorConfig(error);

break;

case 5://mkfile（生成文件）

error = mMkfile();

ErrorConfig(error);

break;

case 6://open（打开文件）

error = mOpen();

ErrorConfig(error);

break;

case 7://close（关闭文件）

error = mClose();

ErrorConfig(error);

break;

case 8://read（读）

error = mRead();

ErrorConfig(error);

break;

case 9://write（写）

error = mWrite();

ErrorConfig(error);

break;

case 10://cp(copy文件)

error = mCp();

ErrorConfig(error);

break;

case 11://rm（删除）

error = mRm();

ErrorConfig(error);

break;

case 12://exit

mExit();

break;

case 13://todo

BackUp();

break;

default:

printf("order not found\n");

break;

}

}

}

Makefile文件：

CC = gcc

all:

$(CC) vfs.c

clean:

rm a.out