****

课程设计报告书

**题目：操作系统课程设计**

**学 院 计算机科学与工程学院**

**专 业 信息安全**

**学生姓名 谢玉萍**

**课程编号 145080**

**课程学分 2.0**

**起始日期 2018-12-1**

**操作系统课程设计**

1. **题目标题和具体要求**
2. 文件系统的操作与编程

1.1文件操作

（1）在~/temp目录下创建名为d1、d2和d3的目录。把文件File1拷贝到 d1目录下，长列表格式显示文件File1，显示的内容包括inode号、 访问权限、硬链接数、文件大小。给出完成这些工作的会话。

（2）在~/temp目录下，把当前目录改变成d2。创建一个名字为 newFile.hard硬链接到d1目录下的File1文件。长列表格式显示 newFile.hard文件，与File1文件的属性进行比较。你如何确定File1 和newFile.hard是同一文件的两个名字，是链接数吗?给出你的会 话过程。

（3）使用硬链接文件newFile.hard显示File1文件的内容。然后取消你本 人对File1文件读(r)权限，再显示文件的内容。

（4）对smallFile文件增加读权限，再一次显示文件内容。

（5） 最后作一个File1文件的备份，并删除File1文件，用newFile.hard显示File1文件内容。

（6）恢复~/temp/d1/File1文件。创建一个名字为~/temp/d2/File.soft

软链接到~/temp/d1/File1文件。长列表格式显示File1.soft文件， 比较这两个文件的属性。你如何确定File1和File.soft是两个不同的 文件?是这两个文件的大小吗?给出会话过程

（7）思考：上述操作执行后文件都发生了什么变化?请了解硬链接文件和软链接文件的概念，解释练习过程中的现象。

1.2文件编程

如下所示目录遍历的关键代码，请理解并扩充程序，使之可运行，实现查看当前目录及子目录的遍历：

char path[PATH\_MAX+1]; DIR \*dp;struct dirent \*dirp;if ((dp = opendir(path)) == NULL) { printf("Cannot read path : %s\n",path); return 0;}while ((dirp = readdir(dp))!=NULL) printf("File name : %s\n",dirp->d\_name);

14 15

closedir(dp)

1.3编写程序

设计并实现一个一级单用户文件系统程序，可提供以下操作:

a. 文件创建/删除接口命令creat/delete(创建的文件不要求格式和内容);

b. 目录创建/删除接口命令mkdir/rmdir;

c. 显示目录内容命令list，要求能够 打印出指定路径或者当前路径下文件的类型(文件或目录)，文件大小，文件 最后修改时间和文件名称。

提示: 使用getcwd，stat，opendir，readdir，closedir等系统调用。

2.文件系统的设计和挑战

2.1分析open，close，read，write这四种对文件的基本操作

在写程序的时候，可能会遇到诸如open，close，read，write等各种对文件的 基本操作，因此在编写代码时只需要直接调用相关的函数。 在linux之中open 操作是由open系统调用完成的，而open系统调用是由函数 sys\_open(fs/open.c)实现的。 同理close，read，write这三种操作也是分别 由sys\_close(fs/open.c)，sys\_read，sys\_write(fs/read\_write.c)系统调用 实现的。 请分析sys\_open，sys\_close，sys\_read，sys\_write这四个函数，并画出具体的流程图。

2.2添加一个文件系统到Linux系统 设计一个类似ext2的自定义文件系统myext2，具体要求如下所述:

2.2.1. myext2文件系统的物理格式定义与ext2基本一致，除了myext2的 magic number是0x6666，而ext2的magic number是0x6666。

2.2.2 myext2是ext2的定制版本，它只支持原来ext2文件系统的部分操

作，以及修改了部分操作。

3.Docker的应用

3.1 Docker基础实验

Docker的安装

Docker的基础命令用法

Docker镜像文件的创建(两种方式)，pull和push

3.2 Docker 应用开发

尝试将以前自己在其他课程(例如C++，Java，数据库，计算机网络 等)所做的实验程序移植到Docker环境

**二、开发工作环境**

操作系统：Linux（Ubuntu）、centos;

编译器：GCC;

程序语言：C

**三、步骤过程**

1.文件系统的操作与编程

1.1文件操作

1.1.1

在~/temp目录下创建名为d1、d2和d3的目录。

输入命令：

parallels@ubuntu:~/temp/d1$ mkdir d1

parallels@ubuntu:~/temp/d1$ mkdir d2

parallels@ubuntu:~/temp/d1$ mkdir d3

在当前目录下新建文件 File1。

parallels@ubuntu:~/temp/d1$ touch File1

把文件File1拷贝到 d1目录下。

parallels@ubuntu:~/temp/d1$ cp ./File1 ./d1/

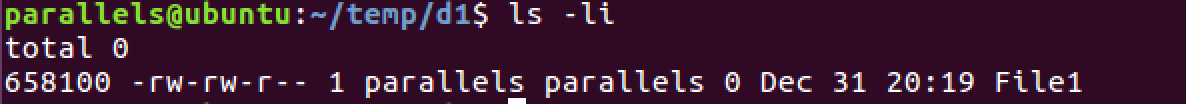
长列表格式显示文件File1，显示的内容包括inode号、 访问权限、硬链接数、文件大小。

parallels@ubuntu:~/temp$ cd d1

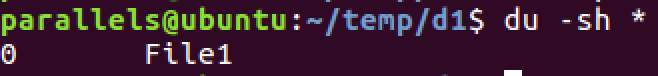
parallels@ubuntu:~/temp/d1$ ls –li 该命令用以查看inode号、 访问权限、硬链接数

parallels@ubuntu:~/temp/d1$ du –sh \* 该命令用以查看文件大小

结果截图：



查看File1的 inode、访问权限和硬连接数



查看 File1的文件大小

文件inode 号为658100，大小为0 byte，硬链接数为1，权限：user和 group有读和写 权限，other 只有读权限。

1.1.2在~/temp目录下，把当前目录改变成d2。创建一个名字为newFile.hard硬链接到d1目录下的File1文件：

parallels@ubuntu:~/temp/d2$ link ../d1/File1 ./newFile.hard



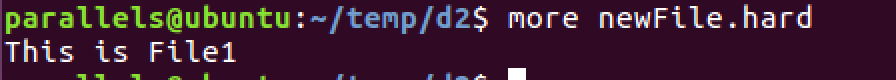
查看两个文件的属性

结论：由于两个文件 inode 相同，故两个文件是同一个文件。并不是根据文件的链接数来判断的。

1.1.3 使用硬链接文件newFile.hard显示File1文件的内容：

parallels@ubuntu:~/temp/d2$ more newFile.hard

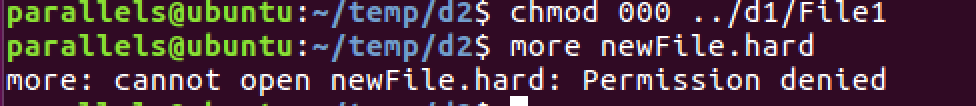
结果截图：



读出的内容与 File1中内容一致

然后取消你本 人对File1文件读(r)权限，再显示文件的内容：

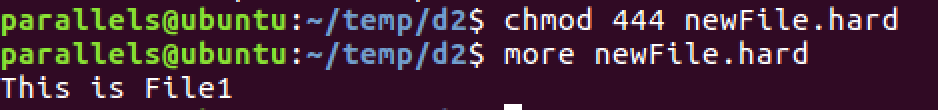
parallels@ubuntu:~/temp/d2$ chmod 000 ../d1/File1



读取 newFile.hard 文件：没有读取权限

1.4.1 对newFile.hard文件增加读权限，再一次显示文件内容

parallels@ubuntu:~/temp/d2$ chmod 444 newFile.hard

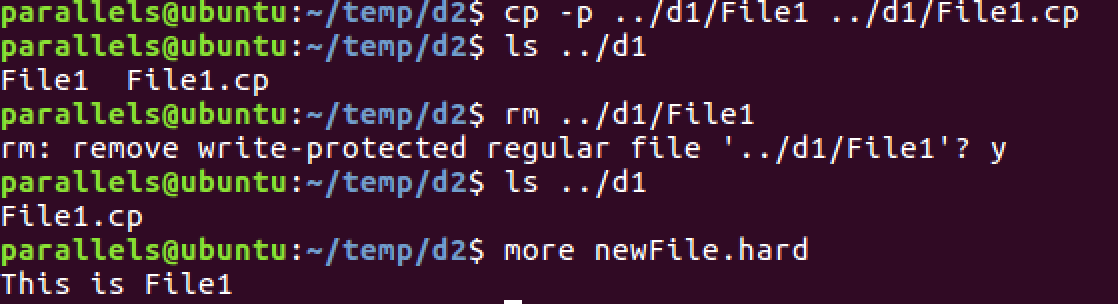


读取 newFile.hard文件：可以读取

1.5.1最后作一个File1文件的备份，并删除File1文件，用newFile.hard显示File1文件内容：输入命令

parallels@ubuntu:~/temp/d2$ cp -p ../d1/File1 ../d1/File1.cp 进行文件备份

parallels@ubuntu:~/temp/d2$ rm ../d1/File1 删除文件



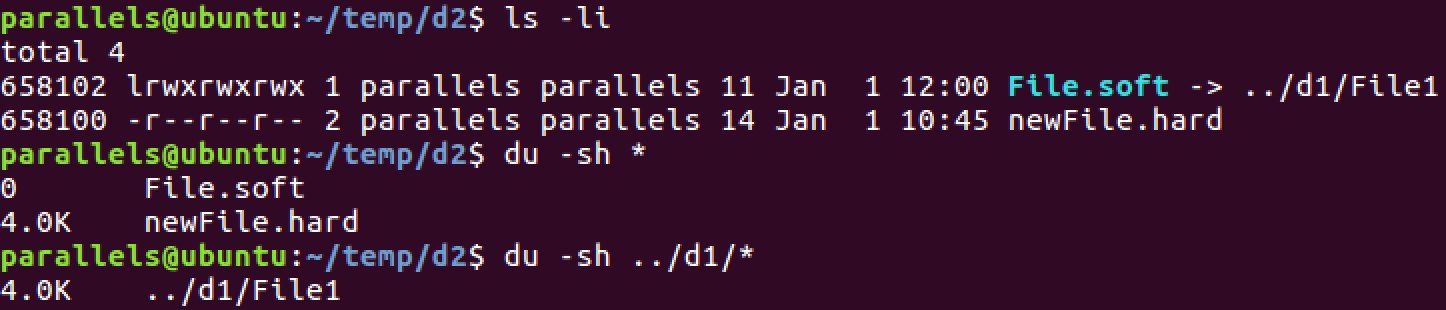
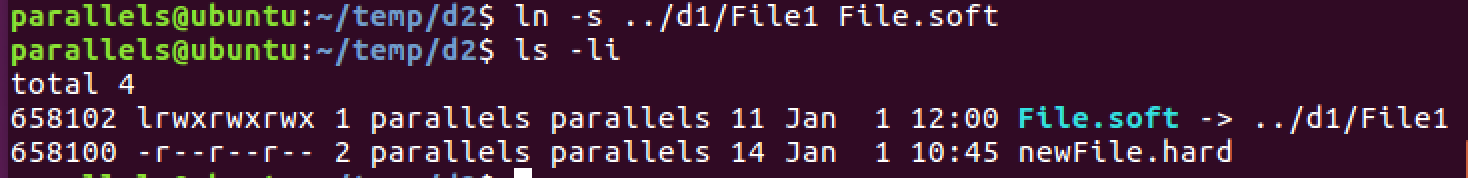
读取 newFile.hard文件

结论：仍可读出文件内容。说明虽然删了 File 文件但数据仍然在内存中没被删除。

1.1.6 恢复~/temp/d1/File1文件。创建一个名字为~/temp/d2/File.soft 软链接到~/temp/d1/File1文件。长列表格式显示File1.soft文件， 比较这两个文件的属性。

输入命令：

parallels@ubuntu:~/temp/d2$ ln -s ../d1/File1 File.soft 建立软连接



你如何确定File1和File.soft是两个不同的文件?是这两个文件的大小吗?

根据File1和 File.oft的 inode 号不同，可判断两个文件不是同一文件。而不是根据文件大小来判断。

1.1.7 思考：上述操作执行后文件都发生了什么变化？请了解硬链接文件和软链接文件的概念，解释练习过程中的现象。

添加硬链接后，两个文件都指向同一块内存空间，文件 inode、权限 和文件大小以及其他属性都是一样的，区别只有文件名不同。所以对其中一个文件进行写操作，实际上是对内存中的数据进行写操作，文件的读操作实际上读取的是同一块内存中的数据，因此两个文件中的内容是同步的。而如果对一个文件的属性进行修改，另一个文件的属性也会同步发生变化，故而修改 File1 的权限也会影响 newFile.hard 的权限。而除非所有硬链接到同一内存地址的文件都被删除，该内存中的数据才会被删除，否则即使删除了其中一些文件，仍然可以通过其他硬链接文件读取该内存中的数据。因此尽管删除了 File1，仍然可通过newFile.hard读出内容。

而添加软连接的文件，文件 inode 和文件大小、文件名与源文件都不一样。软连接是指文件用户数据块中存放的内容是另一文件的路径名的指向，软链接有着自己的 inode 号以及用户数据块。与源文件实际上是不同的文件。

而因而File.soft 与 File1的文件属性不一样。

1.2文件编程

实现查看当前目录及子目录的遍历。

1.2.1定义一个函数 scanDir 进行递归遍历目录，该函数传入的参数为路径名和深度。

该函数先判断路径是否能打开，如果不能，则报错返回；如果能打开，则将返回内容存在DIR 数据结构中。然后使用chdir 函数改变当前工作路径，循环遍历当前目录。如果打开的是仍然是目录，则打印路径并进行递归遍历该目录，并增加深度 depth；如果读取的是文件，则打印该文件名，读取当前目录下的下一个文件或文件夹。一个文件夹循环结束后，返回上级目录继续进行遍历。



1.2.2 测试结果

测试路径 /home/parallels/temp

目录结构

|--File1

|-d1

|--File1

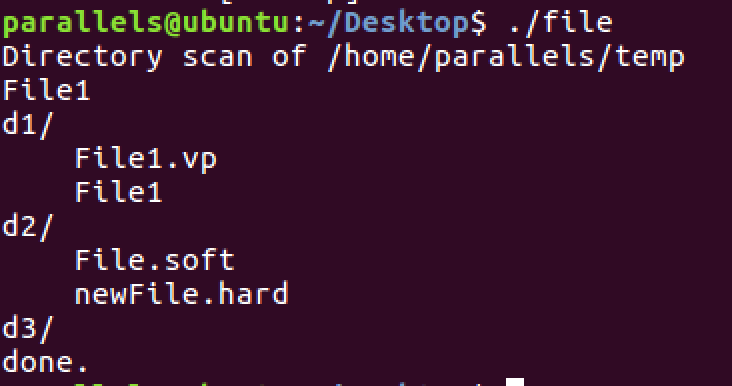
|--File1.vp

|-d2

|--File.soft

|--newFile.hard

|-d3



程序运行结果截图

1.2.3源代码：

#include <unistd.h>

#include <stdio.h>

#include <dirent.h>

#include <string.h>

#include <sys/stat.h>

#include <stdlib.h>

void scanDir(char path[], int depth)

{

DIR \*dp;

struct dirent \*dirp;

struct stat statbuf;

if ((dp = opendir(path)) == NULL) {

printf("Can`t open directory %s\n", path);

return ;

}

chdir(path);

while ((dirp = readdir(dp)) != NULL) {

lstat(dirp->d\_name, &statbuf);

if (S\_ISDIR(statbuf.st\_mode)) {

if (strcmp(dirp->d\_name, ".") == 0 ||

strcmp(dirp->d\_name, "..") == 0 )

continue;

printf("%\*s%s/\n", depth, "", dirp->d\_name);

scanDir(dirp->d\_name, depth+4);

} else

printf("%\*s%s\n", depth, "", dirp->d\_name);

}

chdir("..");

closedir(dp);

}

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

{

char path[120] = "/home/parallels/temp";

printf("Directory scan of %s\n", path);

scanDir(path, 0);

printf("done.\n");

exit(0);

}

1.3编写程序

1.3.1 文件创建/删除：主要用系统调用函数 create 和 remove 实现。

定义函数 creatFile（char\* path）根据传入的文件路径进行文件创建，crat(path,0777)表示创建路径为 path 的文件且文件权限为可读、可写、可执行，文件创建失败返回值为负数；定义函数 deleteFile（char\* path）根据传入的文件路径进行文件删除，remove（path） 函数进行文件删除，如果删除失败返回值为-1。



1.3.2 目录创建/删除：主要使用 mkdir、remove、rmdir

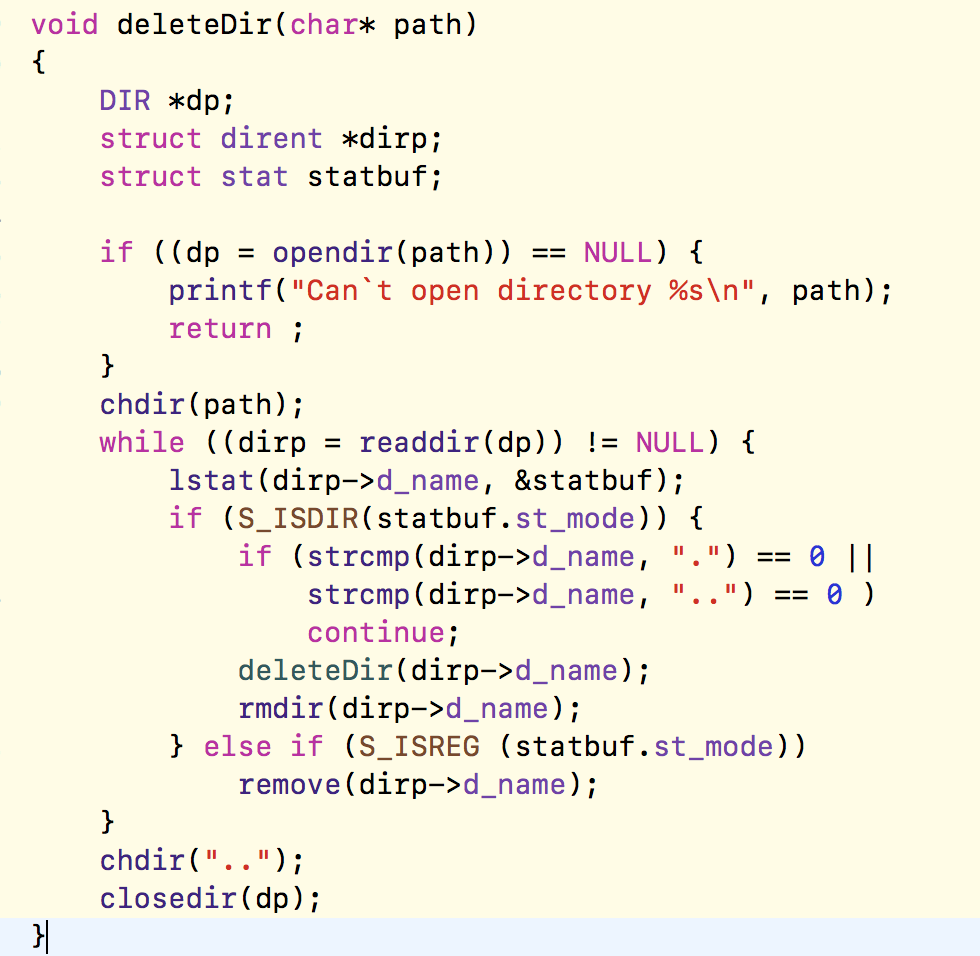
定义函数createDir实现目录的创建，传入参数为路径名。

总体思路是从父到子逐级创建目录：先逐个字符读取路径名，每当遇到’/’时，说明此时为一个目录，先判断当前路径是否存在，如果不存在则创建该目录，再继续往下读取传入的路径名，直至读取完，目录创建成功。



定义函数deleteDir 实现目录的删除。

总体思路：类似于遍历目录，从当前路径出发，在每次读取当前路径后先判断是否为文件，如果是文件则使用 remove 函数删除该文件；如果为文件夹则递归调用 deleteDir 函数，传入该文件夹的路径作为当前目录进行遍历。当一个文件夹内所有文件均已被删除（即文件夹为空）后，使用 rmdir 函数删除该文件夹。

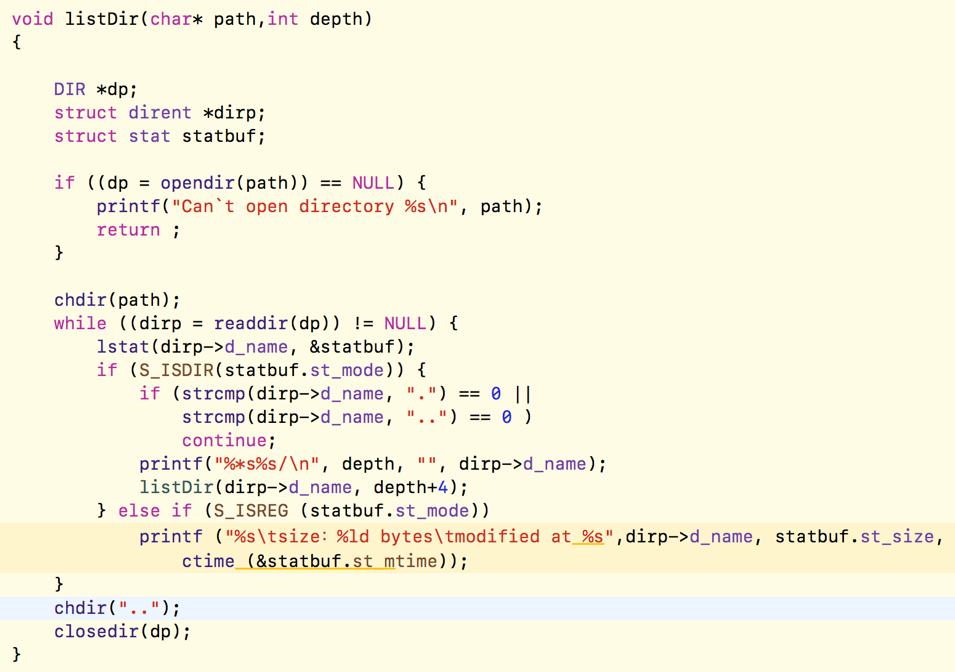


1.3.3显示目录内容命令list，要求能够 打印出指定路径或者当前路径下文件的类型(文件或目录)，文件大小，文件 最后修改时间和文件名称。

定义 listDir 函数实现该功能。

总体思路：类似于目录遍历，只需在打印文件路径名时添加下面函数打印相关信息即可：

printf ("%s\tsize：%ld bytes \tmodified at %s",dirp->d\_name, statbuf.st\_size, ctime (&statbuf.st\_mtime));



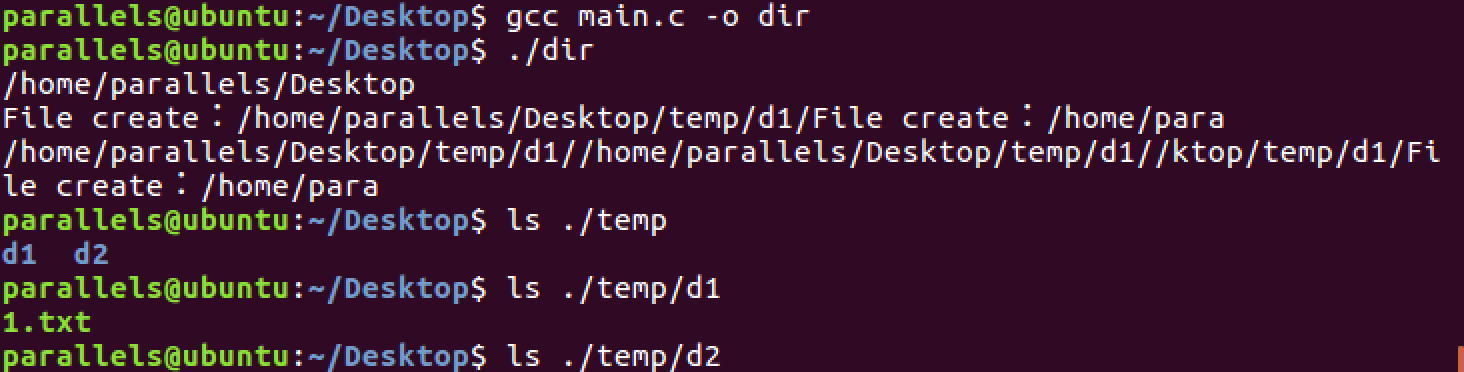
1.3.4程序测试

在当前目录下新建目录./temp/d1 ./temp/d2 和 文件./temp/d1/1.txt。

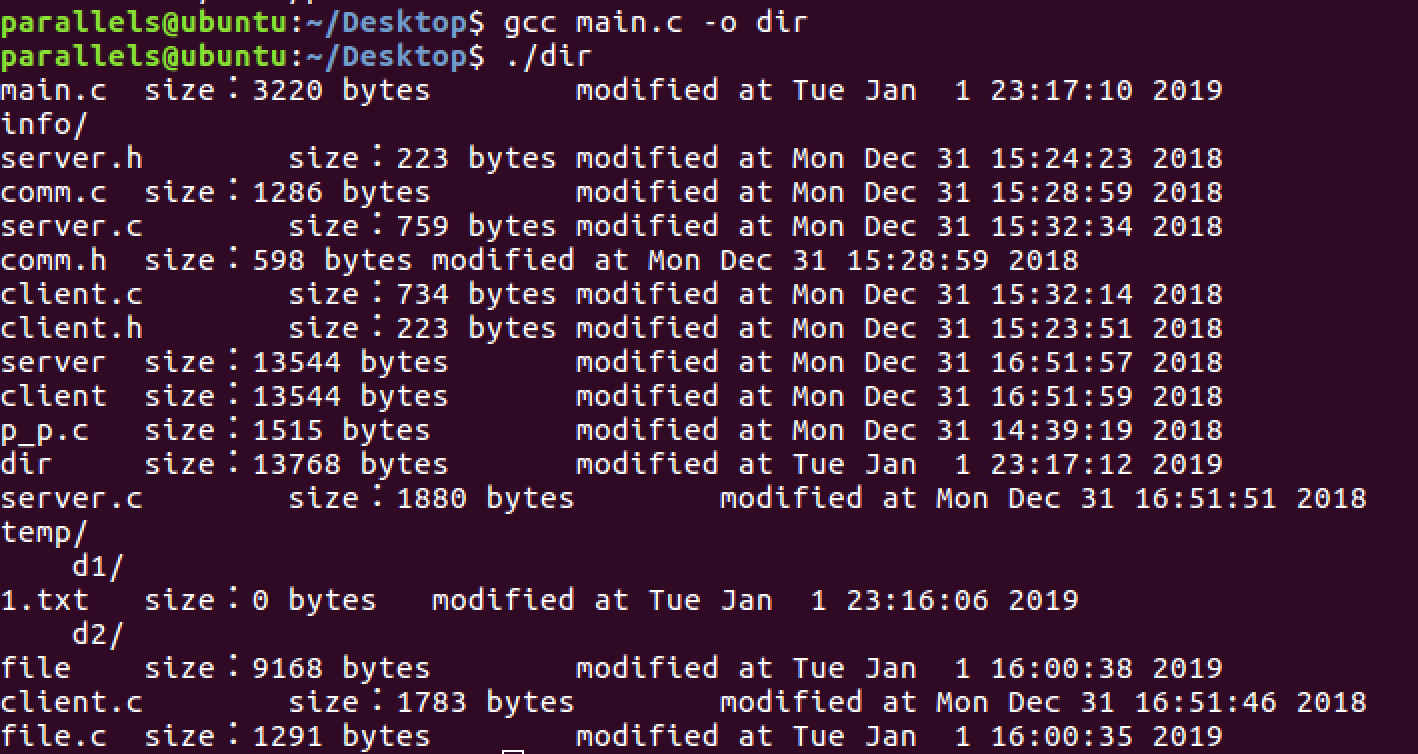
遍历/home/parallels/Desktop目录并打印相关信息。

删除目录temp下所有内容。





创建文件结果截图



遍历目录结果

../Desktop/屏幕快照%202019-01-01%20下午11.19.56.png

删除目录结果

1.3.5 源代码

#include<stdio.h>

#include <fcntl.h>

#include <stdlib.h>

#include <limits.h>

#include <unistd.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <dirent.h>

#include <time.h>

#include <string.h>

void creatFile(char\* path){

if(creat(path,0777)<0)

perror("creat");

else{

printf("File create：%s\n",path);

}

}

void deleteFile(char \* path){

if( remove(path) == -1 )

printf( "File delete:%s\n",path);

else

perror("remove");

}

int creatDir(const char \*path)

{

char DirName[256];

strcpy(DirName, path);

int i,len = strlen(DirName);

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

{

if(DirName[i]=='/')

{

DirName[i] = 0;

if(access(DirName, 0)!=0)

{

if(mkdir(DirName, 0777)==-1)

{

perror("mkdir");

return -1;

}

}

DirName[i] = '/';

}

}

return 0;

}

void listDir(char\* path,int depth)

{

DIR \*dp;

struct dirent \*dirp;

struct stat statbuf;

if ((dp = opendir(path)) == NULL) {

printf("Can`t open directory %s\n", path);

return ;

}

chdir(path);

while ((dirp = readdir(dp)) != NULL) {

lstat(dirp->d\_name, &statbuf);

if (S\_ISDIR(statbuf.st\_mode)) {

if (strcmp(dirp->d\_name, ".") == 0 ||

strcmp(dirp->d\_name, "..") == 0 )

continue;

printf("%\*s%s/\n", depth, "", dirp->d\_name);

listDir(dirp->d\_name, depth+4);

} else if (S\_ISREG (statbuf.st\_mode))

printf ("%s\tsize：%ld bytes\tmodified at %s",dirp->d\_name, statbuf.st\_size, ctime (&statbuf.st\_mtime));

}

chdir("..");

closedir(dp);

}

void deleteDir(char\* path)

{

DIR \*dp;

struct dirent \*dirp;

struct stat statbuf;

if ((dp = opendir(path)) == NULL) {

printf("Can`t open directory %s\n", path);

return ;

}

chdir(path);

while ((dirp = readdir(dp)) != NULL) {

lstat(dirp->d\_name, &statbuf);

if (S\_ISDIR(statbuf.st\_mode)) {

if (strcmp(dirp->d\_name, ".") == 0 ||

strcmp(dirp->d\_name, "..") == 0 )

continue;

deleteDir(dirp->d\_name);

rmdir(dirp->d\_name);

} else if (S\_ISREG (statbuf.st\_mode))

remove(dirp->d\_name);

}

chdir("..");

closedir(dp);

}

int main(void)

{

char\* path = NULL;char\* path2 = NULL;

path = getcwd(NULL,0);

puts(path);

strcat(path,"/temp/");

creatDir(path);

strcat(path,"d1/");

creatDir(path);

strcat(path,"1.txt");

creatFile(path)

path2=getcwd(NULL,0);

puts(path);

strcat(path2,"/temp/d2/");

creatDir(path2);

listDir("/home/parallels/Desktop",0);

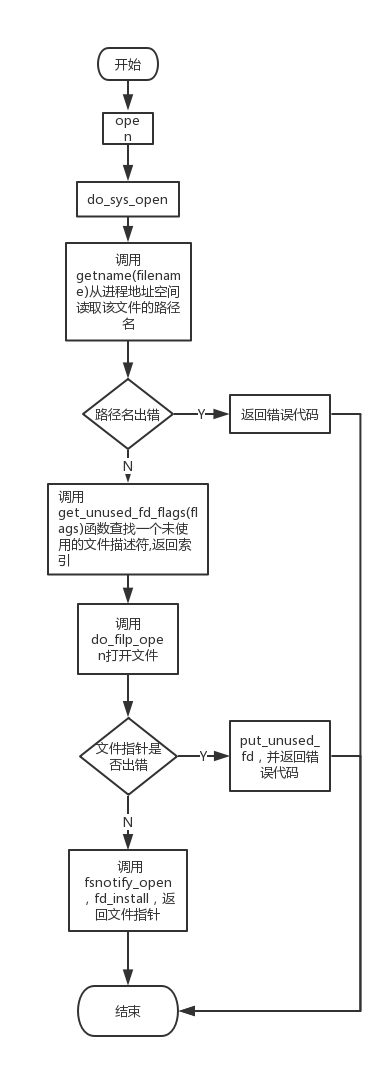
deleteDir("/home/parallels/Desktop/temp");

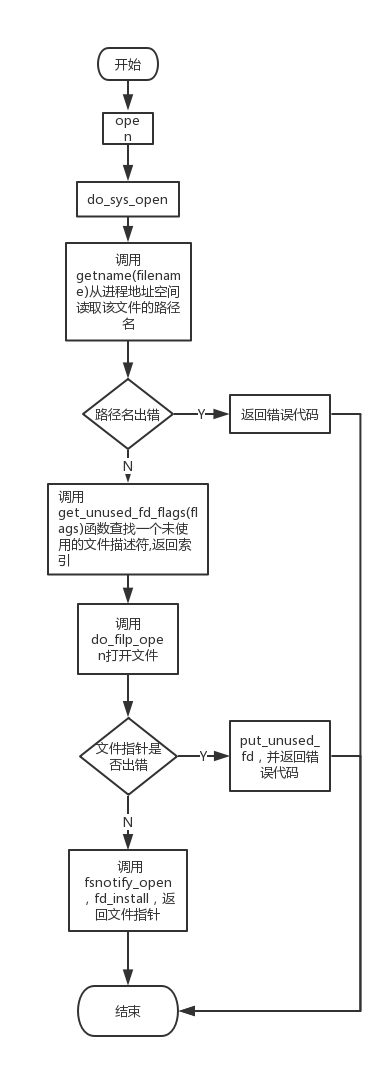
return 0;

}

1. 文件系统的设计和实现（挑战）

2.1分析open，close，read，write这四种对文件的基本操作。分析sys\_open，sys\_close，sys\_read，sys\_write这四个函数，并画出具体的流程图。

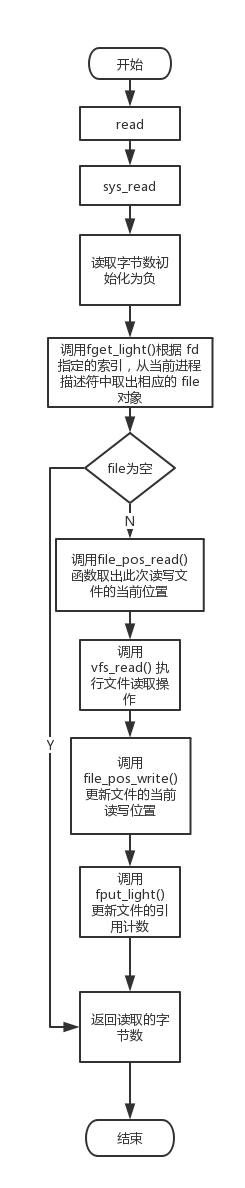
2.1.1 sys\_open



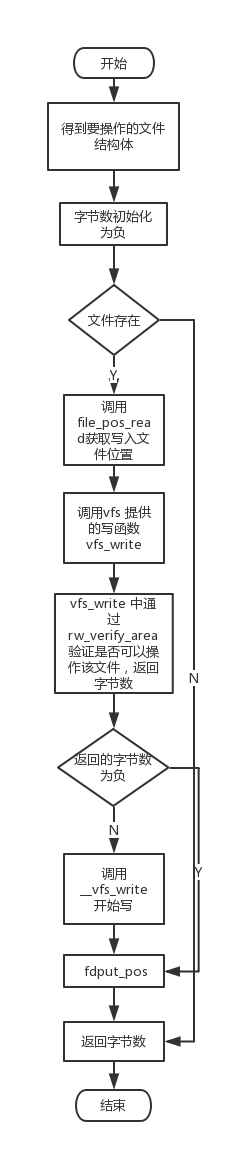
2.1.2 sys\_close



2.1.3 sys\_read

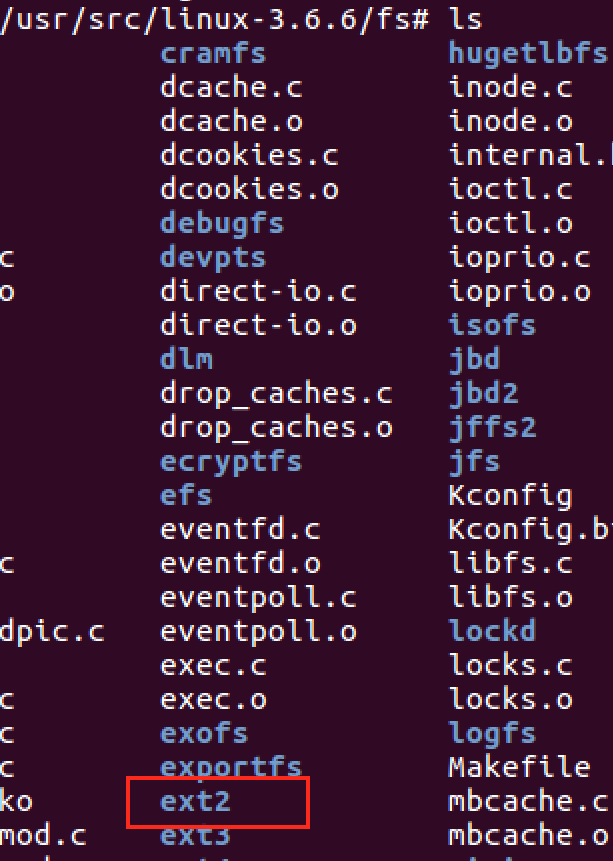


2.1.4 sys\_write



2.2.1下载Linux内核源代码（我是用的版本是3.3.6），并将其解压在/usr/src下。

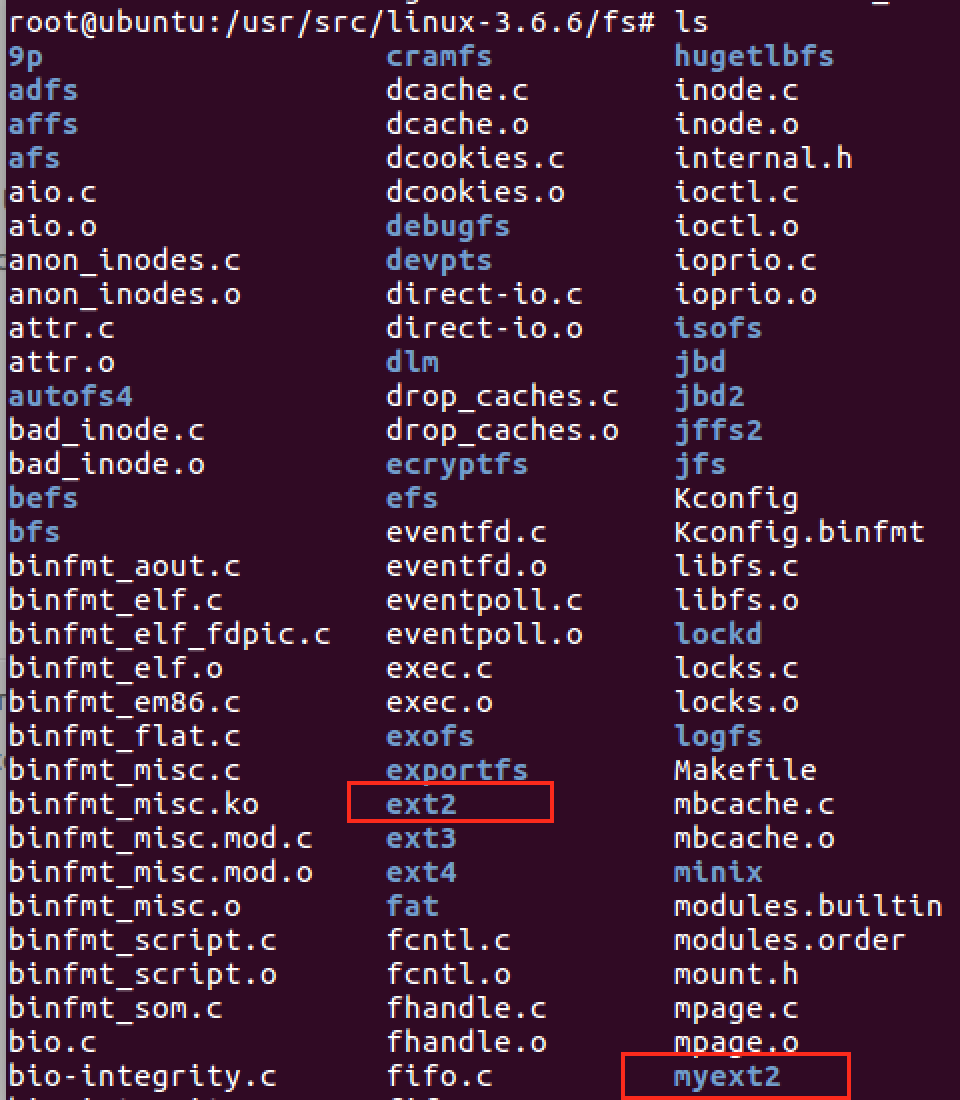
2.2.2在源代码中确认 fs/ext2目录。



ext2目录存在

2.2.3克隆fs/ext2，命名为 fs/myexts。相关命令为：

parallels@ubuntu:/usr/src/linux-3.6.6/fs$ sudo cp -R ext2 myext2



克隆成功

2.2.3.1源代码复制：

$cd ../include/linux

$sudo cp ext2\_fs.h myext2\_fs.h

$cd ../../fs/myext2

$ sudo mv ext2.h myext2.h

$cd ../../include/asm-generic/bitops

$sudo cp ext2-atomic.h myext2-atomic.h

$ sudo cp ext2-atomic-setbit.h myext2-atomic-setbit.h

2.2.3.2 源代码修改：将\*EXT2\*修改为\*MYEXT2\*;将\*ext2\*修改为\*myext2\*。

使用以下脚本替换 fs/myext2中的字符串，并在该目录下建一个名为substitute.sh的可执行文件，然后执行。

substitute.sh文件代码：

#!/bin/bash

SCRIPT=substitute.sh

for f in \*

do

if [ $f = $SCRIPT ]

then

echo "skip $f"

continue

fi

echo -n "substitute ext2 to myext2 in $f..."

cat $f | sed 's/ext2/myext2/g' > ${f}\_tmp

mv ${f}\_tmp $f

echo "done"

echo -n "substitute EXT2 to MYEXT2 in $f..."

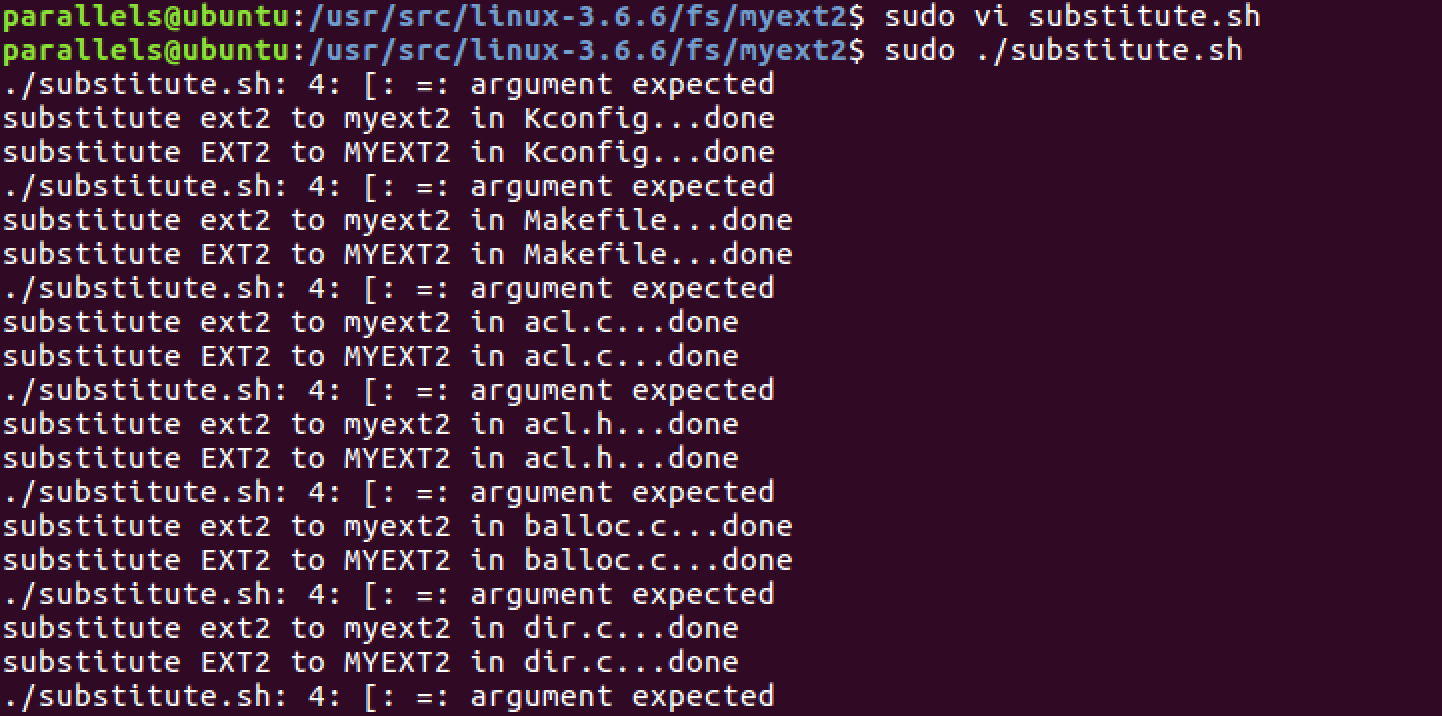
cat $f | sed 's/EXT2/MYEXT2/g' > ${f}\_tmp

mv ${f}\_tmp $f

echo "done"

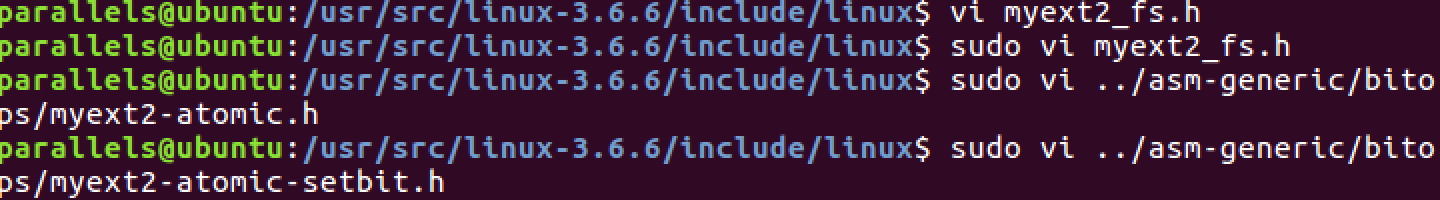
done

执行脚本修改fs/myext2中的字符串：



脚本执行成功

使用 vi 编辑器的文本替换功能（即在命令模式下分别输入%s#ext2#myext2#g、%s#EXT2#MYEXT2#g）将include/linux/myext2\_fs.h 和include/asm-generic/bitops/myext2-atomic.h，include/asm-generic/bitops/myext2-atomic-setbit.h文件中的“ext2”、“EXT2”分别替换成“myext2”、“MYEXT2”



在include/asm-generic/bitops.h文件中添加：

#include <asm-generic/bitops/myext2-atomic.h>

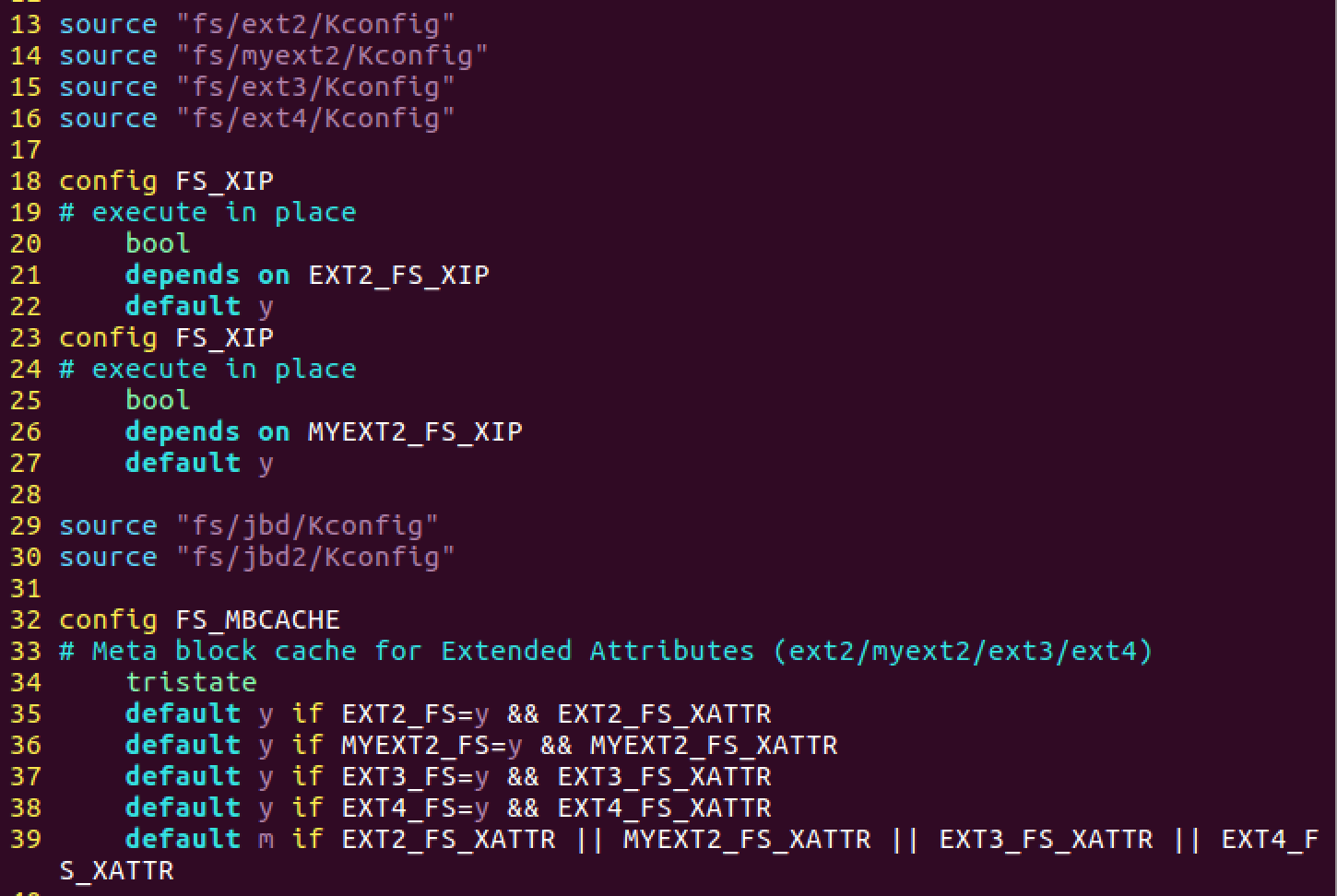
在linux-3.6.6/arch/x86/include/asm/bitops.h文件中添加：

#include <asm-generic/bitops/myext2-atomic-setbit.h>

在include/linux/magic.h中添加：

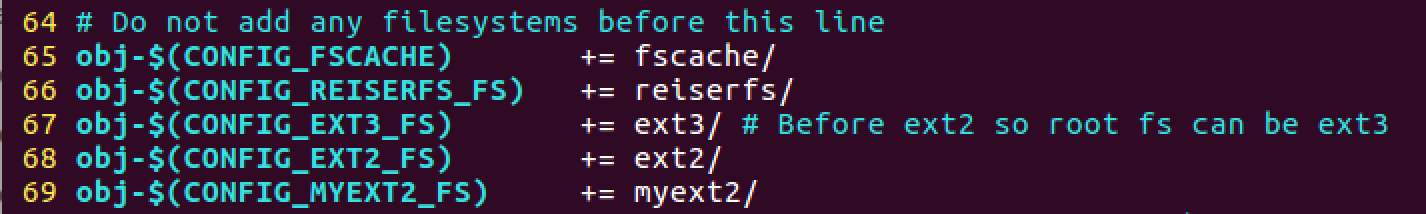
#define MYEXT2\_SUPER\_MAGIC 0xEF53

2.3.3.3 在fs/Kconfig文件中增加 source “fs/myext2/Kconfig”，并且对“ext2”“EXT2”相关项的地方添加“myext2”“MYEXT2”项。



修改 Kconfig 文件

在 fs/Makefile 文件中添加obj-$(CONFIG\_MYEXT2\_FS) += myext2/

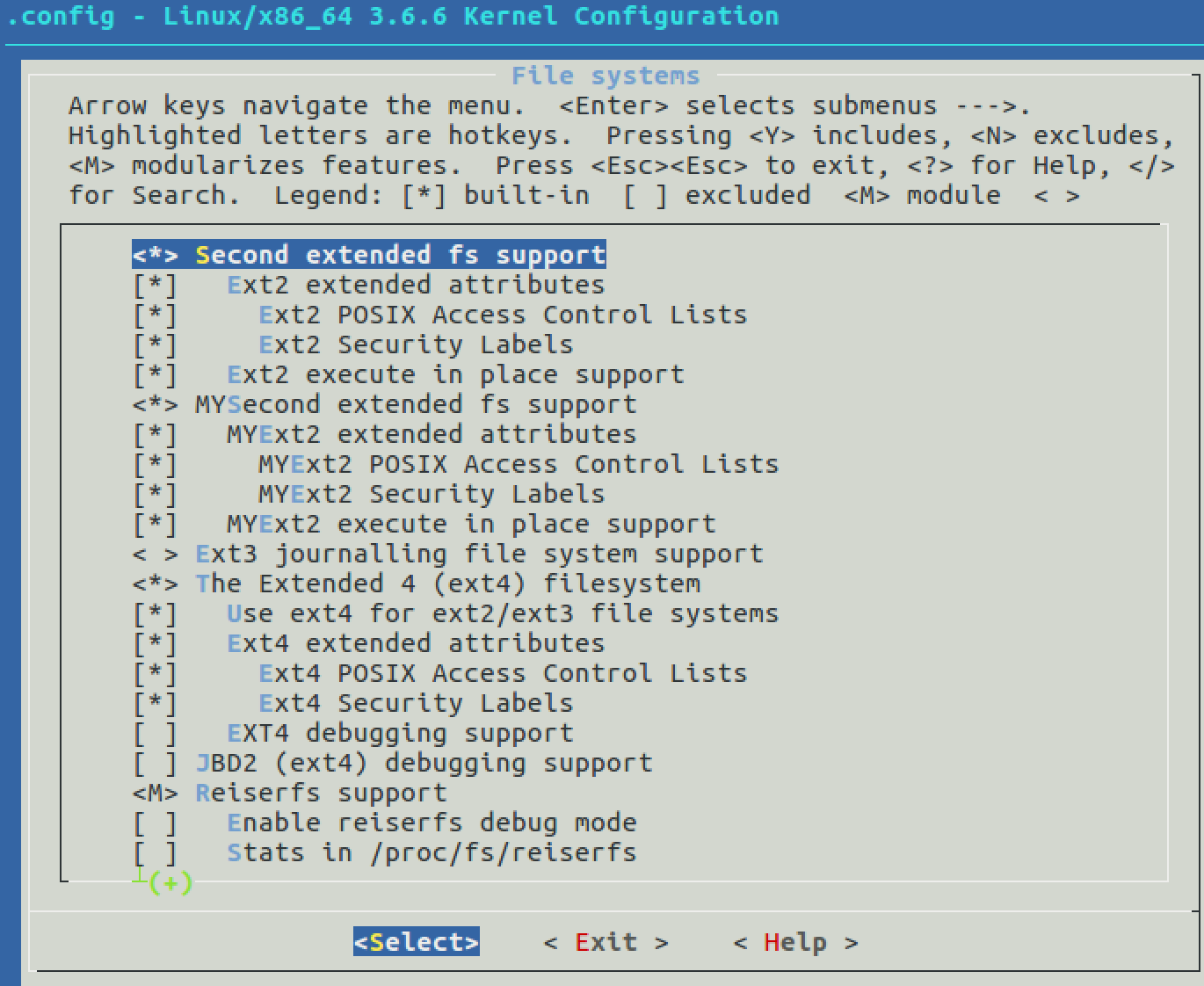


为了在make mencuconfig中看得更加清楚，修改 fs/myext2/Kconfig文件中“Ext2”替换为“MYExt2” ，“Second”前加上“MY”

使用make menuconfig选择上myext2，如下：

# cd /usr/src/linux-3.6.6

# make menuconfig



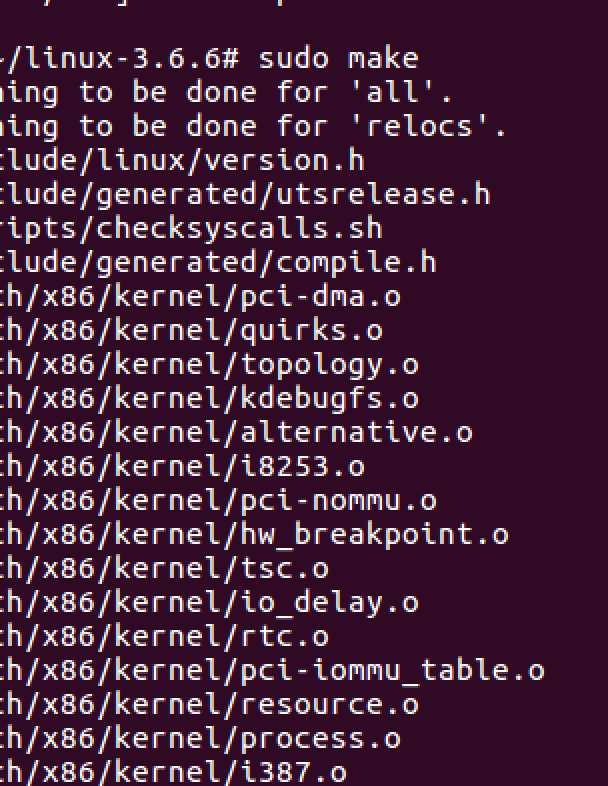
2.2.4编译内核。进入linux-3.6.6/输入以下四个命令

#sudo make

#sudo make modules

#sudo make modules\_install

#sudo make install



对添加的myext2文件系统进行一下测试：

#cd /root

#dd if=/dev/zero of=myfs bs=1M count=1

#/sbin/mkfs.ext2 myfs

#cat /proc/filesystems | grep ext



文件系统配置成功

#sudo mount –t myext2 –o loop ./myfs /mnt

#mount

#sudo umount /mnt

#sudo mount –t ext2 –o loop ./myfs /mnt

#mount



#sudo umount /mnt

修改magci number:在include/linux/magic.h 文件将其改为0x6666，再重新编译内核

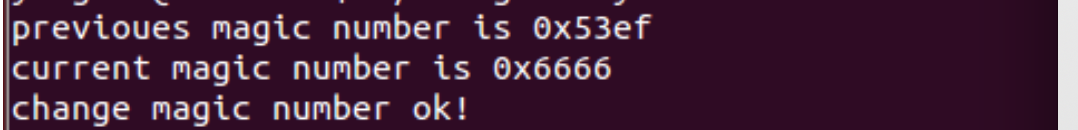
2.2.5 以新的内核启动，使用changeMN.c来修改我们创建的myfs文件系统的magic number。

#gcc –o changeMN changeMN.c 编译该程序

#dd if=/dev/zero of=myfs bs=1M count=1

#/sbin/mkfs.ext2 myfs

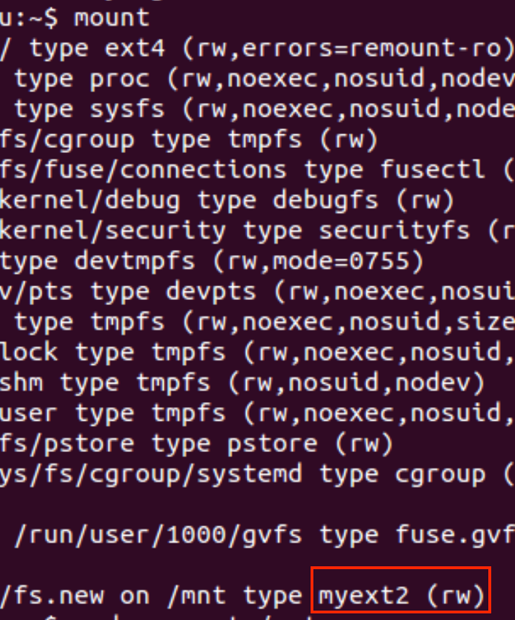
#./changeMN myfs



magic number 修改成功

#sudo mount –t myext2 –o loop ./fs.new /mnt

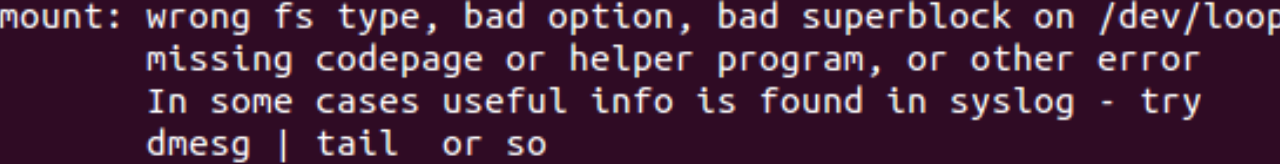
#mount



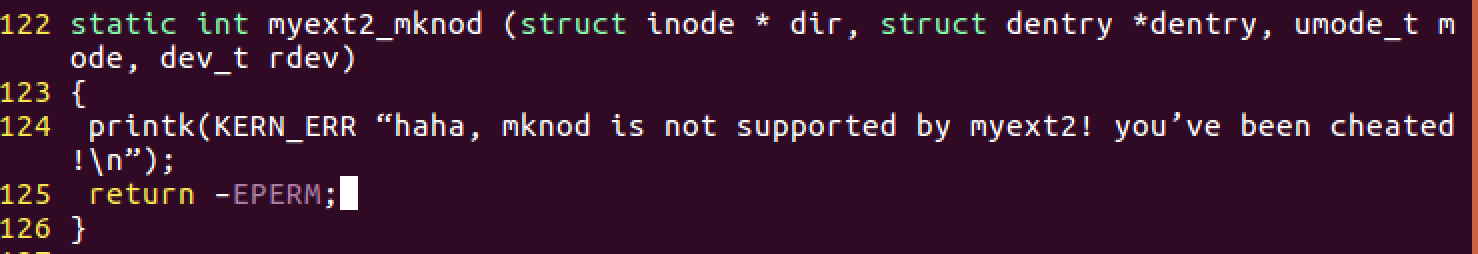
mount myext2成功

#sudo umount /mnt

#sudo mount –t ext2 –o loop ./fs.new /mnt



2.2.6 对fs/myext2/namei.c的mknod函数作如下修改：



第124行:打印信息来说明mknod操作不被支持。

第125行:将错误号为EPERM的结果返回给shell，即告诉shell，在myext2文件系统中，maknod不被支持。

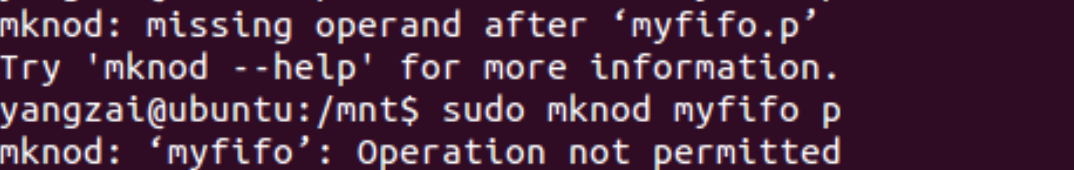
2.2.7 修改完毕后重新编译内核。以新生成的内核重新启动计算机，在shell下执行如下测试程序：

# sudo mount –t myext2 –o loop ./fs.new /mnt 将fs.new mount到/mnt目录下。

#cd /mnt 进入fs.new这个myext2文件系统。

# sudo mknod myfifo p 执行创建一个名为myfifo的命名管道的命令。

运行结果：



挂载失败

#dmesg |tail 打印出错信息:

../Desktop/屏幕快照%202019-01-02%20下午5.09.08.png

3.Docker的应用

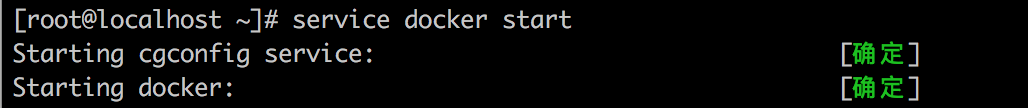
3.1 Docker的安装

输入命令：

#yum install -y epel-release

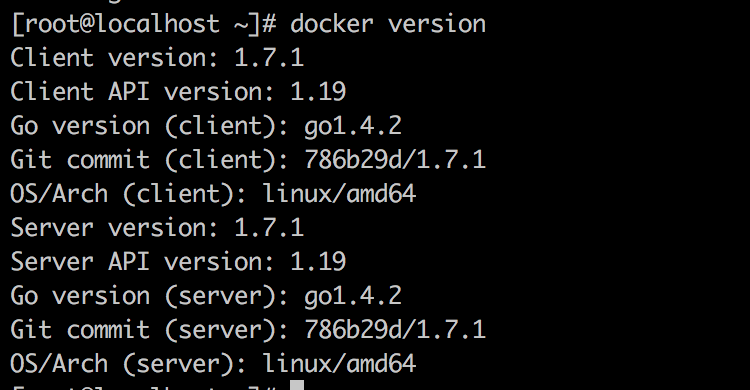
#yum install docker-io # 安装docker

#service docker start # 启动docker服务

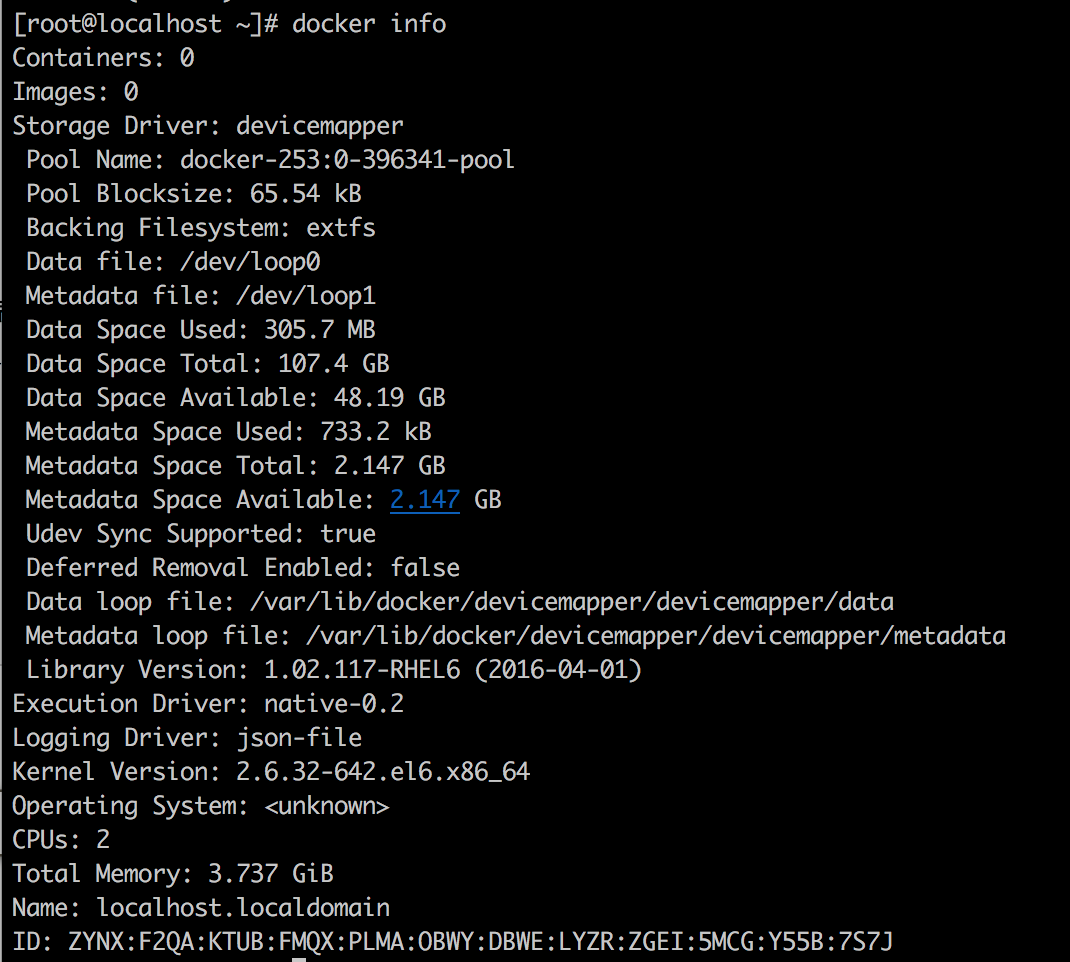


3.2 Docker 基础命令

#docker version 查看版本号、



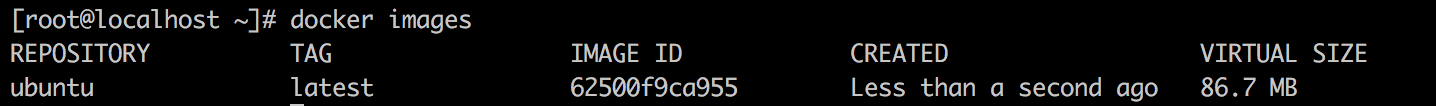
#docker info 查看系统（docker）层面信息



3.3 Docker 镜像文件的创建

#docker pull ubuntu 下载 Ubuntu镜像

#docker images 查看镜像



**四、测试或验证结果**

实验过程以及结果要截图，对实验结果进行必要的解释说明。

1.文件编程测试结果

测试路径 /home/parallels/temp

目录结构

|--File1

|-d1/

|--File1

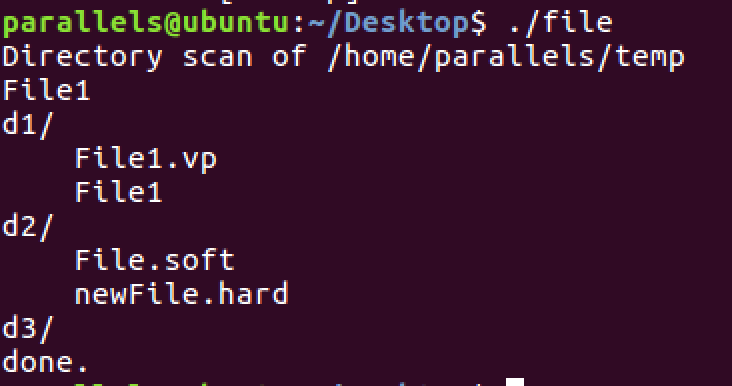
|--File1.vp

|-d2/

|--File.soft

|--newFile.hard

|-d3



程序运行结果截图

结论：成功遍历目录。

2.编写程序测试结果

测试路径 ~/Desktop

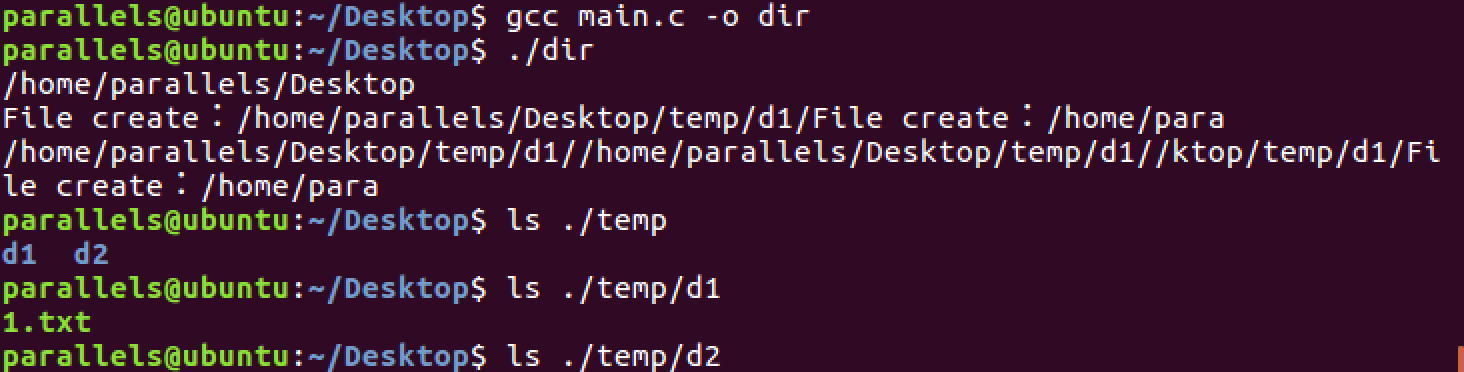
新建目录的结构

|-temp/

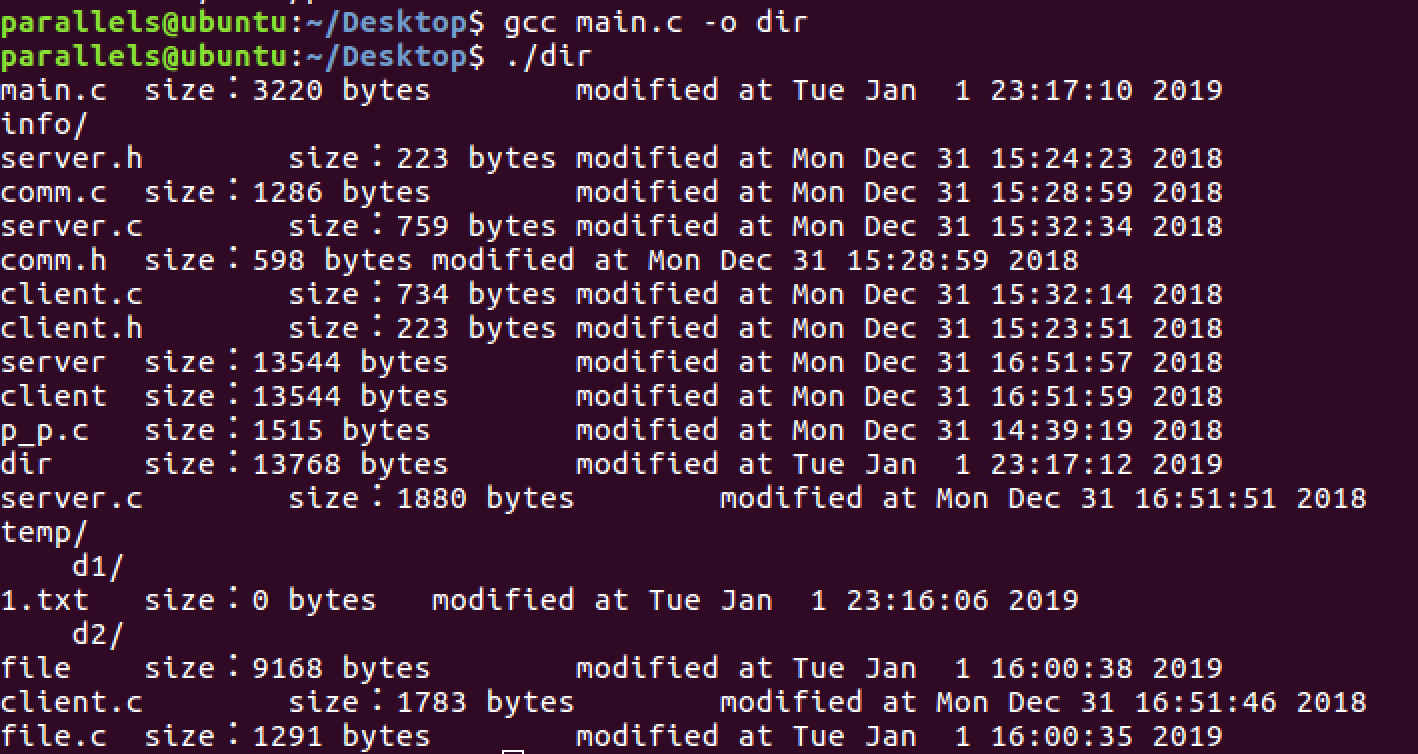
|--d1/

|---1.txt

|--d2/



创建目录和文件结果截图



遍历目录结果

../Desktop/屏幕快照%202019-01-01%20下午11.19.56.png

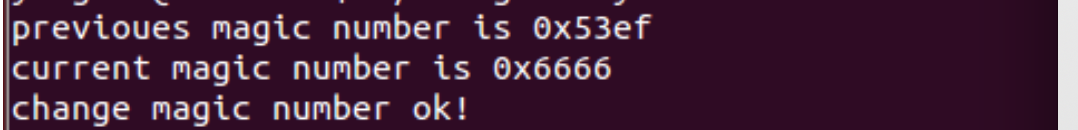
删除文件结果

结论：成功创建/删除文件，创建/删除目录，打印文件/目录大小、修改时间。

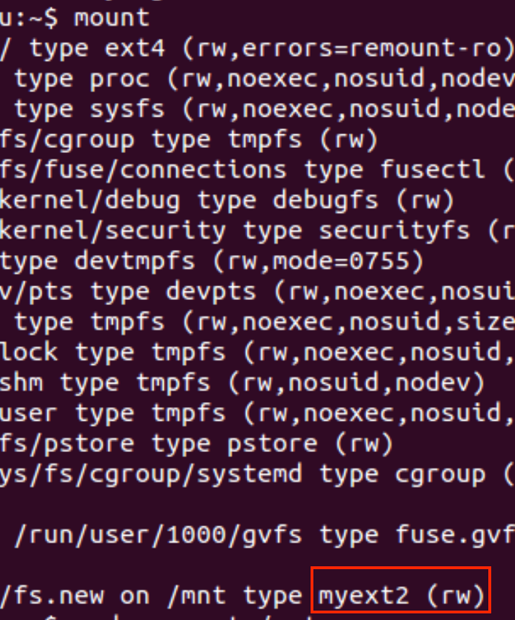
3.文件系统的实现和挑战



文件系统配置成功

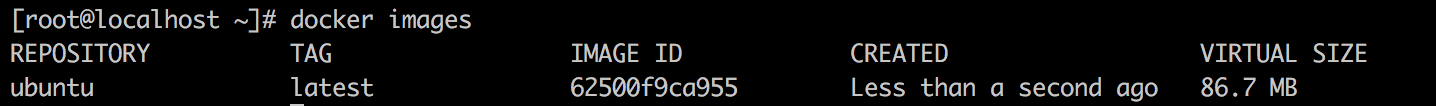


magic number 修改成功



创建并mount 新的 myext2文件分区成功

4.Docker应用开发



创建镜像文件成功

**五、工作小结**

在本次课程设计中，我成功地完成了文件系统的操作与编程以及文件系统的设计和挑战，也在这个过程中获益匪浅。

通过本次课程设计，我对于软链接和硬链接的概念、实质、区别和使用有了更加深刻的了解，对于文件的额访问权限之间的关系也更加清晰明了。而在完成文件编程和编写程序后，我对于一些有关文件操作的系统调用函数也有了一定的了解，并且学会了如何对目录进行创建、遍历、删除等操作。分析了 read，open，write，close 调用过程后，我对这些函数的使用更加得心应手，也知道了更多底层的关于文件系统的知识。通过对文件系统的设计和实现，我更加深入地了解了操作系统中文件系统的原理和实现，对于 Linux 中的 ext2文件系统也有了更多的体会。

参考文献

［1］Andrew S.Tanenbaum，Herbert Bos. 现代操作系统［M］．北京：机械工业出版社，2018. 04.

［2］浙江大学远程教育学院．添加一个文件系统［EB/0L］．http://jpkc.scezju.com/czxtyl/redir.php?catalog\_id=105547.2018-12-28．