**课 程 实 验 报 告**

**课程名称： 操作系统**

**实验名称： Linux文件系统**

**专业班级： 计算机Ⅱ类2103班**

**学 号： 8008121077**

**姓 名： 陈映融**

**指导教师： 胡勇**

**报告日期： 2023/06/20**

## 1.1 实验概述

实验目的：通过使用Linux的文件和目录的API，增加对存储设备及文件系统的了解，并能创建文件系统的实用工具

实验目标：

1. 完成mystat工具要求使用stat()函数接口打印文件大小、分配的块数、引用计数。
2. 编写myls工具使用stat()、opendir()、readdir()和getcwd()接口 来实现ls的功能，传入-l参数时也同ls -l一样能打印目录的具体信息包括但不限于出文件的所有者，所属组权限，修改时间以及stat()函数获取的信息。

实验要求：通过验证实验内容掌握 API，解决实验设计问题，提交报告。

实验语言：c

实验环境：linux、gcc

## 1.2 实验过程及结果

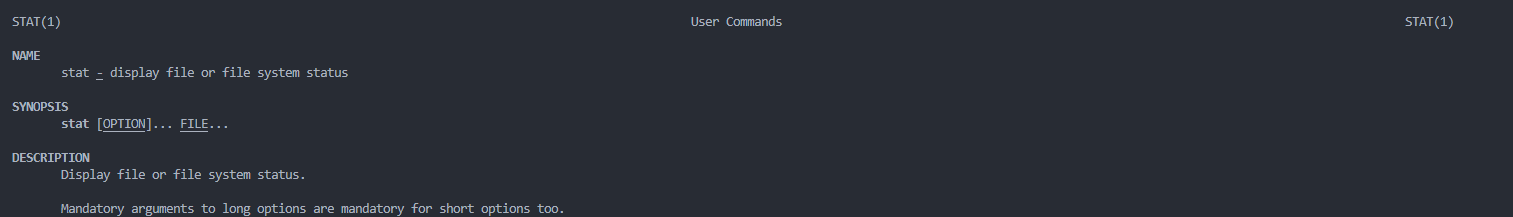
**1.2.1 mystat文件系统工具**

实验要求：Linux 提供的 stat 程序，请编写您自己版本的命令行程序 mystat，实现对给定文件或目录进行信息读取分析，并打印出文件大小、分配 的块数、引用（链接）计数等。【接口：stat()】

stat()函数

int stat(const char \*path, struct stat \*dir);

man手册是怎么描述stat函数的



显示文件或文件系统状态。

长期权的强制性参数对于短期权也是强制性的。

stat()函数有两个参数，一个是文件的路径，第二格式stat的结构体指针。传入结构体dir就可以把打开路径的文件相关信息。

stat()函数的返回值：成功获取文件信息时，返回值为0。如果调用stat()函数失败，返回值为-1，并设置相应的错误码。它会通过设置全局变量errno的信息来表示错误信息。

stat结构体的定义如下

struct stat

{

    dev\_t     st\_dev;         *// 文件的设备ID*

    ino\_t     st\_ino;         *// 文件的i-node节点号*

    mode\_t    st\_mode;        *// 文件的类型和权限*

    nlink\_t   st\_nlink;       *// 连接到该文件的硬链接数目*

    uid\_t     st\_uid;         *// 文件所有者的用户ID*

    gid\_t     st\_gid;         *// 文件所有者的组ID*

    dev\_t     st\_rdev;        *// 如果文件是特殊文件，则保存设备ID*

    off\_t     st\_size;        *// 文件的大小（以字节为单位）*

    time\_t    st\_atime;       *// 文件的最后访问时间*

    time\_t    st\_mtime;       *// 文件的最后修改时间*

    time\_t    st\_ctime;       *// 文件的最后状态改变时间*

    blksize\_t st\_blksize;     *// 文件系统的I/O缓冲区大小*

    blkcnt\_t  st\_blocks;      *// 分配给文件的512字节块的数量*

};

好了有了以上知识对于实验1.2.1的难度仅限于结构体的访问和排版。以下是我写的mystat.c的代码

#include <stdio.h>

#include <sys/stat.h>

#include <unistd.h>

#include <assert.h>

#include <time.h>

#include <pwd.h>

#include <grp.h>

int main(int argc, char \*argv[])

{

    if (argc < 2)

    {

        printf("参数不足\n用法: mystat <file or path>\n");

        return 1;

    }

    char \*path = argv[1];

    struct stat file;

    assert(stat(path, &file) == 0);

    struct passwd \*pw = getpwuid(file.st\_uid);

    struct group \*gr = getgrgid(file.st\_gid);

    printf("File: %s\n", path);

    printf("Size: %ld \t\t", file.st\_size);

    printf("Blocks: %ld\n", file.st\_blocks);

    printf("Inode: %ld\t", file.st\_ino);

    printf("Links: %ld\n", file.st\_nlink);

    printf("Access: %o\t", file.st\_mode & 0777);

    printf("Uid:( %d/%s)\t", file.st\_uid, pw->pw\_name);

    printf("Gid:( %d/%s)\n", file.st\_gid, gr->gr\_name);

    printf("ACCESS: %s", ctime(&file.st\_atime));

    printf("Modify: %s", ctime(&file.st\_mtime));

    printf("Change: %s", ctime(&file.st\_ctime));

    return 0;

}

stat在shell中的运行结果

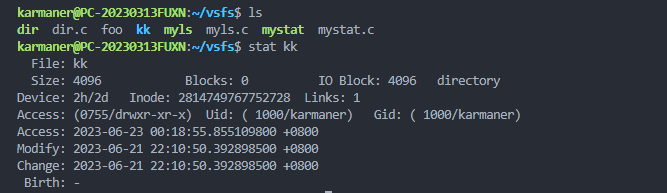


图 1 vsfs的文件格式和Linux的stat格式

由图可知vsfs文件有dir，myls， mystat三个可执行文件，kk一个目录文件，和其他四个普通文件。

以下是mystat描述的文件

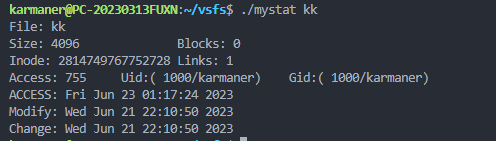


图 2 mystat工具对文件的描述

我的mystat有缺陷比如文件类型没有文件类型，其实只要对st\_mode进行处理就可以的出文件类型没有进行转化处理是因为时间有限。通过对pw，gr两个结构体对uid和gid的解析，解析出拥有者名字，和所属组。

**1.2.2 myls工具**

实验要求：编写一个命令行程序 myls（如同 Linux 提供的 ls 程序），列出指定目录内容。如果没有传参数，则程序仅输出指定目录中的文件名。当传入 -l 参数时，程序需要打印出文件的所有者，所属组权限以及 stat() 函数获得的一些其他信息。另外还要支持传入要读取的目录作为参数，比如 myls -l directory。如果没有传入目录参数，则用当前目录作为默认参数。【接口：stat()、opendir()、readdir()和getcwd()】

老规矩我们先看看man手册对于ls的描述

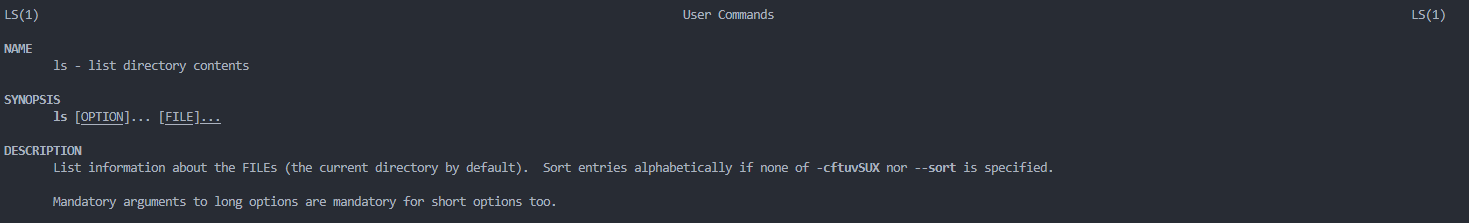


图 3 man手册中的ls

man手册中的ls -l描述



函数接口

opendir():

opendir()函数用于打开一个目录，返回一个指向DIR类型的指针，以便后续对目录进行操作。

#include <sys/types.h>

#include <dirent.h>

DIR \*opendir(const char \*dirname);

opendir()返回值：

如果成功打开目录，则返回一个指向DIR类型的指针，该指针可用于后续的目录操作。

如果打开目录失败，则返回NULL，并设置errno来指示错误类型。

getcwd():

getcwd()函数用于获取当前工作目录的路径。

#include <unistd.h>

char \*getcwd(char \*buf, size\_t size);

返回值：

如果成功获取当前工作目录的路径，则返回指向缓冲区的指针。

如果发生错误，则返回NULL，并设置errno来指示错误类型。

readdir():

readdir()函数用于读取打开目录后的文件列表。它从一个已打开的目录流中读取下一个目录项。

#include <dirent.h>

struct dirent \*readdir(DIR \*dirp);

dirp：指向已打开目录的指针（使用opendir()函数获得）

返回值：

如果成功读取到下一个目录项，则返回指向struct dirent结构体的指针。

如果到达目录流的末尾或者发生错误，则返回NULL。

struct dirent的结构如下：

struct dirent

{

    ino\_t          d\_ino;       *// inode number*

    off\_t          d\_off;       *// offset to the next dirent*

    unsigned short d\_reclen;    *// length of this record*

    unsigned char  d\_type;      *// type of file*

    char           d\_name[];    *// filename*

};

有了以上知识我现在要做的工作就只有三部分

1. 对myls -l path 和myls的两种模式区分开
2. myls输出结果
3. myls -l path输出结果
4. 改进-l的位置可以在1位置也可以在2位置
5. 尽量 和ls的界面靠拢

处理两种模式通过main函数传入的argc统计的传入参数个数来实现argc > 1才是能读到文件的路径和参数，确定路径的位置如果-l在2位置那么dir\_path = argv[1] 如果-l在1位置那么dir\_path = argv[2],以下是我选择参数函数的代码

char \*dir\_path = "."; *// 默认使用当前目录*

    int long\_format = 0;  *// 是否使用长格式输出*

*// 检查命令行参数*

    if (argc > 1)

    {

        if (strcmp(argv[1], "-l") == 0)

        {

            long\_format = 1;

            if (argc > 2)

            {

                dir\_path = argv[2];

            }

        }

        else if (strcmp(argv[1], "-l") != 0 && argc > 2 && strcmp(argv[2], "-l") == 0)

        {

            long\_format = 1;

            dir\_path = argv[1];

        }

        else

        {

            dir\_path = argv[1];

        }

    }

选择参数后即选择路径以后(显示格式也选择了)就可以对目录的内容进行处理。处理myls和myls -l path or myls path -l

第一部就是打开目录

*// 打开目录*

    DIR \*dir = opendir(dir\_path);

    if (dir == NULL)

    {

        perror("Failed to open directory");

        return 1;

    }

*// 获取当前工作目录*

    char cwd[PATH\_MAX];

    if (getcwd(cwd, sizeof(cwd)) == NULL)

    {

        perror("Failed to get right working directory");

        closedir(dir);

        return 1;

    }

*// 切换到指定目录*

    if (chdir(dir\_path) != 0)

    {

        perror("Failed to change directory");

        closedir(dir);

        return 1;

    }

其实核心的代码也就三条

DIR \*dir = opendir(dir\_path);  *// 打开目录*

getcwd(cwd, sizeof(cwd));      *// 获取当前工作目录*

chdir(dir\_path);               *// 切换到指定目录*

其余的都是对异常的处理。其中dir\_path在路径获取阶段已经完成。

在第二部完成以后就是对内容的打印myls只需要打印文件，myls -l需要打印具体信息。我的设计如下：

*// 读取目录内容*

    struct dirent \*entry;

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

    {

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

        {

            continue;

        }

        if (long\_format)

        {

*// 获取文件信息*

            struct stat fileStat;

            if (stat(entry->d\_name, &fileStat) == 0)

            {

                print\_file\_info(entry->d\_name, &fileStat);

            }

            else

            {

                perror("Failed to get file information");

            }

        }

        else

        {

*// 仅输出文件名*

            printf("%s  ", entry->d\_name);

        }

    }

*// 换行*

    printf("\n");

*// 关闭目录*

    closedir(dir);

如果是long\_format则通过print\_file\_info()函数来打印具体的文件信息。如果不是就只答应名称。下面我们来具体进入函数print\_file\_info()函数

void print\_file\_info(const char \*filename, const struct stat \*fileStat)

{

    struct passwd \*pw = getpwuid(fileStat->st\_uid);

    struct group \*gr = getgrgid(fileStat->st\_gid);

*// 打印文件权限*

    printf((S\_ISDIR(fileStat->st\_mode)) ? "d" : "-");

    printf((fileStat->st\_mode & S\_IRUSR) ? "r" : "-");

    printf((fileStat->st\_mode & S\_IWUSR) ? "w" : "-");

    printf((fileStat->st\_mode & S\_IXUSR) ? "x" : "-");

    printf((fileStat->st\_mode & S\_IRGRP) ? "r" : "-");

    printf((fileStat->st\_mode & S\_IWGRP) ? "w" : "-");

    printf((fileStat->st\_mode & S\_IXGRP) ? "x" : "-");

    printf((fileStat->st\_mode & S\_IROTH) ? "r" : "-");

    printf((fileStat->st\_mode & S\_IWOTH) ? "w" : "-");

    printf((fileStat->st\_mode & S\_IXOTH) ? "x" : "-");

*// 打印硬链接计数*

    printf(" %2ld", fileStat->st\_nlink);

*// 打印文件所有者和所属组*

    printf(" %8s %8s", pw->pw\_name, gr->gr\_name);

*// 打印文件大小*

    printf(" %9ld", fileStat->st\_size);

*// 打印修改时间*

    char timebuf[80];

    strftime(timebuf, sizeof(timebuf), "%b %d %H:%M", localtime(&fileStat->st\_mtime));

    printf(" %s", timebuf);

*// 打印文件名*

    printf(" %s\n", filename);

}

现在代码都已经完成我们来看运行结果

Linux shell的ls



图 4 shell ls格式

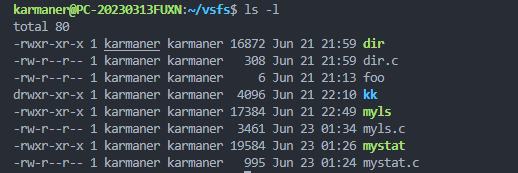


图 5 shell ls -l的格式

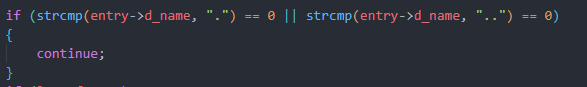


图 6 ls -l 对用户目录的打印

我的ls工具

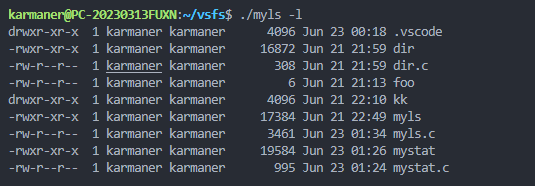


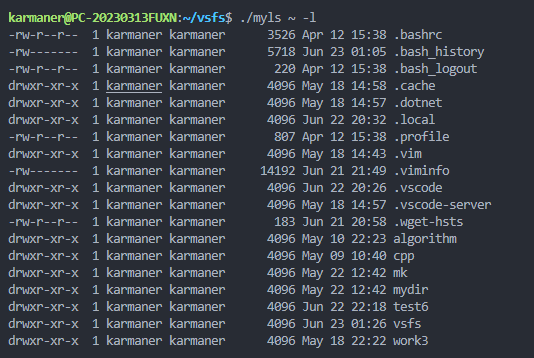
可以看到我们把隐藏文件读出来了，其是我是有对.本目录文件 ..父目录文件进行隐藏处理如下



我的mysl -l工具实验结果







可以看到把一些本该隐藏的文件显示出来了.vim文件vim的配置文件，.vscode vscode的配置文件等等。

## 1.3 实验小结及心得

在本次实验中我学习了一些与Linux目录操作相关的关键函数。您了解了如何打开目录、读取目录中的文件列表以及关闭目录即目录相关的API和文件相关的API。

1. 使用opendir()函数打开目录：学会了使用opendir()函数打开目录，通过提供目录路径作为参数，并获得一个指向DIR类型的指针，以便后续对目录进行操作。
2. 使用readdir()函数读取目录项：学会了使用readdir()函数从已打开的目录中读取下一个目录项。该函数返回一个指向struct dirent结构体的指针，包含有关目录项的信息，例如文件名等。
3. 处理目录项：通过使用readdir()函数，可以循环读取目录中的文件列表。也可以根据需要处理每个目录项的信息，例如打印文件名、进行过滤或其他操作。
4. stat描述的文件的信息的处理，比如对st\_uid、st\_gid、st\_mode等文件信息的处理

本次myls的程序和ls程序的相似度有90%但是对于其他功能-a等是没有任何实现。本次实验让我认识到通过这些实验，我对Linux中的目录操作有了更深入的了解，并掌握了基本的目录遍历和文件列表处理技巧。这些技能在编写需要处理文件系统的应用程序或脚本时非常有用。

我会继续加强对Linux系统的学习和实践，将进一步提升您的编程能力和对操作系统的理解。