Linux作业2

> 专业院系:计算机学院计算机科学与技术系
> 班级: 2015211306班
> 魏晓: 2015211301

实验目的

- 练习vi，使用Linux的系统调用和库函数，体 会Shell文件通配符的处理方式以及命令对选项的处理方式

实验内容

- 编程实现程序list.c，列表普通磁盘文件（不考虑目录和设备文件等），列出文件名和文件大小。
- 与ls命令类似，命令行参数可以有0到多个
    - 0个参数：列出当前目录下所有文件
    - 参数为普通文件：列出文件
    - 参数为目录：列出目录下所有文件
- 实现自定义选项r,a,l,h,m以及--
    - r 递归方式列出子目录
    - a 列出文件名第一个字符为圆点的普通文件（默认情况下不列出 文件名首字符为圆点的文件）
    - l 后跟一整数，限定文件大小的最小值（字节）
    - h 后跟一整数，限定文件大小的最大值（字节）
    - m 后跟一整数n，限定文件的最近修改时间必须在n天内
    - -- 显式地终止命令选项分析

实验代码

```c
#include <stdio.h>  // printf
#include <stdlib.h>  // malloc/free

#include <time.h>  // time
#include <string.h>  // strcmp, strlen

#include <errno.h>  // errno
#include <sys/types.h>  // for xxx_t

#include <unistd.h>  // getopt
#include <getopt.h>  // getopt (fallback)

#include <dirent.h>  // for DIR, dirent, opendir, readdir, closedir
#include <sys/stat.h>  // for stat

typedef struct Opts
{
    const char *argv0;//local file
    int recurFlag; //deter recursive
    int allFlag; //deter all (including leading with dot)
    off_t LowSize; //the lower bound of size;
    off_t highSize; //the upper bound of size;
    time_t modiTime; //modify time in seconds;
}opt;

void output(const opt *pOpts, const char *pName, int firstFlag);

int main(int argc, char **argv)
{
    opt os1={ 0 };

    os1.argv0=argv[0];
    {
        int temp=0;
        while(-1 != (temp=getopt(argc, argv, "ral:h:m:")))
            switch (temp) {
                case 'r':
                    os1.recurFlag=1;
                    break;
                case 'a':
                    os1.allFlag=1;
                    break;
                case 'l':
                    os1.LowSize=atoi(optarg);
```

```
                    break;
                case 'h':
                    os1.highSize=atoi(optarg);
                    break;
                case 'm':
                    os1.modiTime=atoi(optarg)* 24*60*60;
                    break;
                default:
                    break;
            }
    }

    size_t   argcOpt = optind <argc? argc- optind :1;

    const char **argvOpt = malloc (sizeof(const char *)* argcOpt);
    {
        int i=0;
        while(optind<argc)
            argvOpt[i++] = argv[optind++];
        if(!i)
            argvOpt[i++]=".";
    }

    for (int i=0;i<argcOpt;i++)
    {
        if(argcOpt!=1&& i!=0)
            printf("\n");
        output(&os1,argvOpt[i],1);
    }

    free(argvOpt);
    return 0;
}

void output(const opt * pOpts, const char *pName, int firstFlag)
{
    struct stat status;

    if(stat(pName,&status)==-1)
    {
        fprintf(stderr,"%s (stat):cannot access \'%s\':%s\n",pOpts->argv0,pName
,strerror(errno));
        return;
    }
```

```
    const mode_t stat_mode =status.st_mode;
    const off_t stat_size = status.st_size;
    const time_t stat_mtime = status.st_mtime;

    if (S_ISREG (stat_mode))
    {
        // Test mTime, lSize, hSize
        char regFlag =
        (!pOpts->modiTime || time (NULL) - stat_mtime <= pOpts->modiTime) &&
        (!pOpts->LowSize || stat_size >= pOpts->LowSize) &&
        (!pOpts->highSize || stat_size <= pOpts->highSize);

        if (regFlag || firstFlag)
        {
            char timeStr[64];
            strftime (timeStr, 64, "%m-%d %H:%M", localtime (&stat_mtime));
            printf ("%s \n(%lld byte) (%s)\n\n", pName, stat_size, timeStr);
        }
    }
    else if (S_ISDIR (stat_mode))
    {
        // NOT Test mTime, lSize, hSize
        printf ("%s\n", pName);

        // Open this directory
        DIR *pDir = opendir (pName);
        if (pDir == NULL){
            fprintf (stderr, "%s (opendir): cannot access \'%s\': %s\n",
                     pOpts->argv0, pName, strerror (errno));
            return;
        }
        // Traverse this directory
        struct dirent *pDirEntry;
        while (NULL != (pDirEntry = readdir (pDir)))// judge the folder
        {
            const char *pEntryName = pDirEntry->d_name;
            // Test aFlag
            if (!pOpts->allFlag && pEntryName[0] == '.')
                continue;
            // Test special directory
            if (strcmp (pEntryName, ".") == 0 || strcmp (pEntryName, "..") == 0
)//if the last folder or the next folder is null.continue
                continue;
            // Construct new name
            size_t len1 = strlen (pName);
```

```
            size_t len2 = strlen (pEntryName);
            char *newName = malloc (len1 + 1 + len2 + 1);
            //malloc memory contain pname and pentryname
            strcpy (newName, pName);
            newName[len1] = '/';
            strcpy (&newName[len1 + 1], pEntryName);
            newName[len1 + 1 + len2] = 0;
            // Test rFlag, firstFlag
            if (pOpts->recurFlag || firstFlag)//recursive of go continue
                output(pOpts, newName, 0);//find files in son folder
            free (newName);//free the memory of malloc
        }
        closedir (pDir);//free the memory og dir
    }
}
```

实验结果

用法:

```
list [-a] [-r] [-l] [-h] [-m <day>] [path ...]
```

编译:

```
Xiaos-MacBook-Pro:linuxlist weixiao$ gcc -Wall -g -o list main.c
```

效果:

```
Xiaos-MacBook-Pro:linuxlist weixiao$ ./list -r /Users/weixiao/Documents/Xcode/linuxlist/linuxlist
/Users/weixiao/Documents/Xcode/linuxlist/linuxlist
/Users/weixiao/Documents/Xcode/linuxlist/linuxlist/list.dSYM
/Users/weixiao/Documents/Xcode/linuxlist/linuxlist/list.dSYM/Contents
/Users/weixiao/Documents/Xcode/linuxlist/linuxlist/list.dSYM/Contents/Resources
/Users/weixiao/Documents/Xcode/linuxlist/linuxlist/list.dSYM/Contents/Resources/DWARF
/Users/weixiao/Documents/Xcode/linuxlist/linuxlist/list.dSYM/Contents/Resources/DWARF/list
(14977 byte) (05-03 21:06)

/Users/weixiao/Documents/Xcode/linuxlist/linuxlist/list.dSYM/Contents/Info.plist
(633 byte) (05-03 21:06)

/Users/weixiao/Documents/Xcode/linuxlist/linuxlist/main.c
(5640 byte) (04-29 11:39)

/Users/weixiao/Documents/Xcode/linuxlist/linuxlist/list
(9952 byte) (05-03 21:06)
```