lab04Copy One Directory

软件工程 2018 级 1813075 刘茵

Target

1. Write a c/c++ program

2. To implement copy one diretory and it's subdiretories with multi-threads

3. Gcc

---

1) Install GCC Software Colletion

```
>> sudo apt-get install build-essential
```

2) How to use GCC

- gcc and make

3) Struct of directory

```
struct dirent
{
ino_t d_ino; //d_ino 此目录进入点的 inode
ff_t d_off; //d_off 目录文件开头至此目录进入点的位移
signed short int d_reclen; //d_reclen _name 的长度, 不包含 NULL 字符
unsigned char d_type; //d_type d_name 所指的文件类型 d_name 文件名
har d_name[256];
};

opendir()
readdir()
closedir()
```

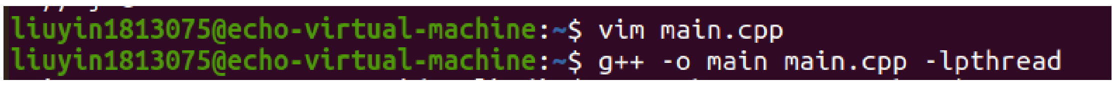
4) Write a c program to implement copy one diretory and it's subdiretories, and the program also verifies the result

- 编写 main.cpp 文件:

```
#include <iostream>
#include <string>
#include <unistd.h>
#include <sys/stat.h>
#include <dirent.h>
#include <pthread.h>
#include <semaphore.h>
#include <stack>
#include <cstring>
#include <fcntl.h>
#include <cstdlib>
#include <vector>
#include <sys/wait.h>

#define NUM_OF_THREADS 10
using namespace std;
sem_t my_sem;

struct FileOperation
{
    string src_file;
    string dst_file;
};
int times = 0;
stack<FileOperation> homework;

//复制文件
void cp(string src_file, string dst_file)
{
    int fd_read = open(src_file.c_str(), O_RDONLY, S_IREAD | S_IWRITE | S_IRGRP | S_IROTH);
    int fd_write = open(dst_file.c_str(), O_WRONLY | O_CREAT, S_IREAD | S_IWRITE | S_IRGRP | S_IWGRP | S_IROTH);
    cout << dst_file.c_str() << endl;
    cout << src_file.c_str() << endl;
    cout << fd_read << ":" << fd_write << endl;
    cout << src_file << ":" << dst_file << endl;
    if (fd_read == -1 || fd_write == -1)
    {
        cout << "when copy " << src_file << "复制失败 !" << endl;
        cout << "failed:" << times << endl;
```

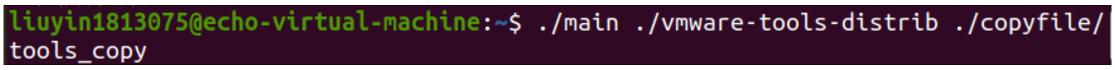
```cpp
        times++;
    }
    else
    {
        char buf[1024];
        int size = 0;
        while ((size = read(fd_read, buf, 1024)) > 0)
        {
            write(fd_write, buf, size);
        }
        cout << "file:" << src_file << "复制成功 !" << endl;
    }
    close(fd_write);
    close(fd_read);
    cout << "总失败次数" << times << endl;
}

//遍历目录
void walk_dir(const char *src_dir, const char *dst_dir)
{
    struct dirent *filename;
    DIR *dir;
    dir = opendir(src_dir); //获得目录信息
    if (dir == NULL)
    {
        cout << "打开 src_dir 失败" << endl;
        exit(0); //结束当前进程
    }
    cout << "open src_dir success!" << endl;
    char path[256];
    while ((filename = readdir(dir)) != NULL)
    { //读取文件夹下文件
        if ((!strcmp(filename->d_name, ".")) || (!strcmp(filename->d_name, "..")))
        {
            continue; //遇到. ..就跳过
        }
        snprintf(path, 256, "%s/%s", src_dir, filename->d_name); //按照"%S"格式储存到 path 中
        struct stat s;
        lstat(path, &s); //获取 path 详细信息储存进 s
        if (S_ISDIR(s.st_mode))
        { // S_ISDIR()函数 判断一个路径是否为目录
            char sub_src_dir[256];
            char sub_dst_dir[256];
            snprintf(sub_dst_dir, 256, "%s/%s", dst_dir, filename->d_name);
            snprintf(sub_src_dir, 256, "%s/%s", src_dir, filename->d_name);
            cout << sub_dst_dir << endl;
            mkdir(sub_dst_dir, S_IWUSR | S_IRUSR | S_IXUSR | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH); //!!竟然把 dst 写成了 src
            walk_dir(sub_src_dir, sub_dst_dir);
        }
        else
        { //是文件
            char dst_file[256];
            char src_file[256];
            snprintf(dst_file, 256, "%s/%s", dst_dir, filename->d_name);
            snprintf(src_file, 256, "%s/%s", src_dir, filename->d_name);
            struct FileOperation new_operation;
            new_operation.src_file = src_file;
            new_operation.dst_file = dst_file;
            homework.push(new_operation);
        }
    }
    closedir(dir);
}

void *run(void *)
{
    struct FileOperation opreation;
    while (!homework.empty())
    {
        sem_wait(&my_sem);
        opreation.dst_file = homework.top().dst_file;
        opreation.src_file = homework.top().src_file;
        homework.pop();
        sem_post(&my_sem); //释放锁
        cp(opreation.src_file, opreation.dst_file);
    }
    return NULL;
}
int main(int argc, char *argv[])
{
    sem_init(&my_sem, 0, 1);
    if (argc < 3)
    {
        cout << "please give right path" << endl;
        exit(0);
    }
    struct stat s;
    //检查文件夹是否有校
    lstat(argv[1], &s);
    if (!S_ISDIR(s.st_mode))
    {
        cout << "the source path is wrong" << endl;
        exit(0);
    }
    //检查目标文件夹是否有效;
    lstat(argv[2], &s);
    if (!S_ISDIR(s.st_mode))
    {
        cout << "the dest path is wrong" << endl;
    }
    walk_dir(argv[1], argv[2]);
    vector<pthread_t> threads;
    threads.resize(NUM_OF_THREADS); //设置线程数目
    for (int i = 0; i < threads.size(); i++)
    {
        pthread_create(&threads[i], NULL, run, NULL);
    }
    for (int i = 0; i < threads.size(); i++)
    {
        pthread_join(threads[i], NULL);
    }
    return 0;
}
```

- 编译并运行多线程 main.cpp

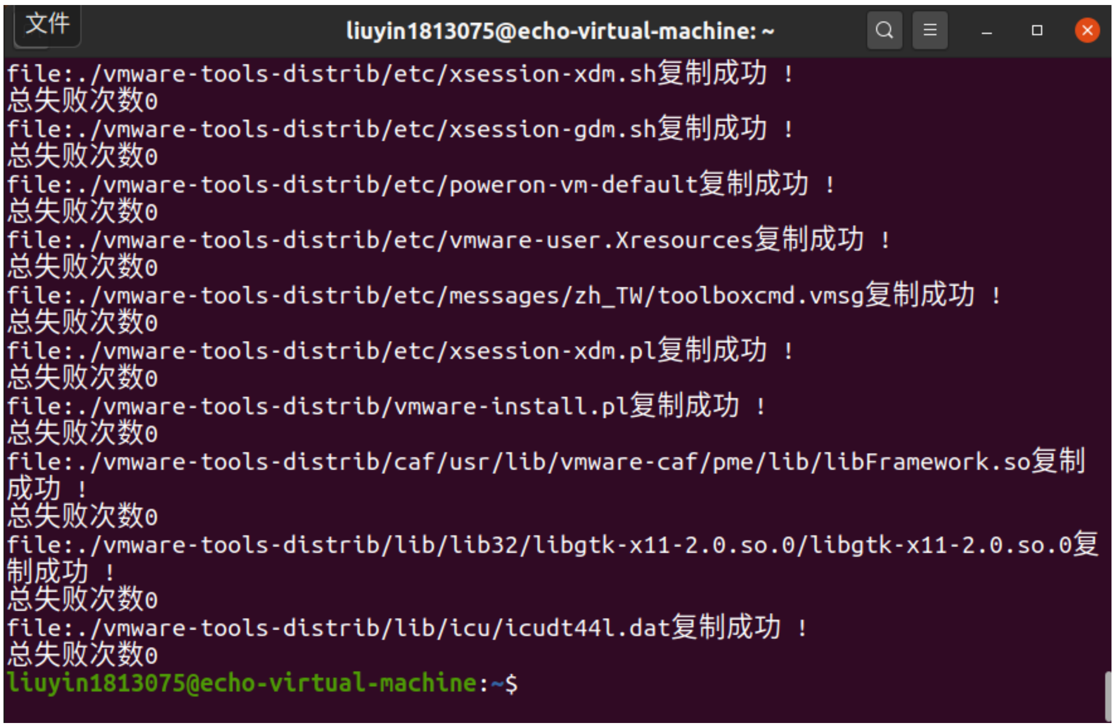


```
liuyin1813075@echo-virtual-machine:~$ vim main.cpp
liuyin1813075@echo-virtual-machine:~$ g++ -o main main.cpp -lpthread
```

- 将 vmware-tools-distrib 文件夹及其子文件夹下内容拷贝



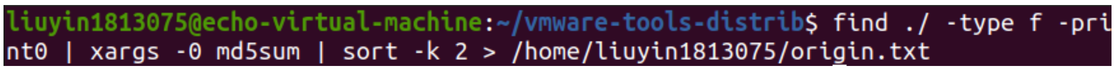
```
liuyin1813075@echo-virtual-machine:~$ ./main ./vmware-tools-distrib ./copyfile/
tools_copy
```



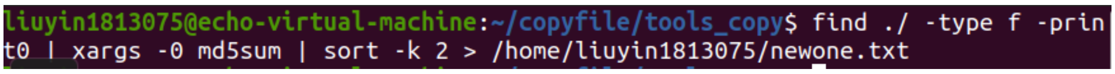
```
file:./vmware-tools-distrib/etc/xsession-xdm.sh复制成功 !
总失败次数0
file:./vmware-tools-distrib/etc/xsession-gdm.sh复制成功 !
总失败次数0
file:./vmware-tools-distrib/etc/poweron-vm-default复制成功 !
总失败次数0
file:./vmware-tools-distrib/etc/vmware-user.Xresources复制成功 !
总失败次数0
file:./vmware-tools-distrib/etc/messages/zh_TW/toolboxcmd.vmsg复制成功 !
总失败次数0
file:./vmware-tools-distrib/etc/xsession-xdm.pl复制成功 !
总失败次数0
file:./vmware-tools-distrib/vmware-install.pl复制成功 !
总失败次数0
file:./vmware-tools-distrib/caf/usr/lib/vmware-caf/pme/lib/libFramework.so复制
成功 !
总失败次数0
file:./vmware-tools-distrib/lib/lib32/libgtk-x11-2.0.so.0/libgtk-x11-2.0.so.0复
制成功 !
总失败次数0
file:./vmware-tools-distrib/lib/icu/icudt44l.dat复制成功 !
总失败次数0
liuyin1813075@echo-virtual-machine:~$
```

(输出成功提示)

- 将源文件夹和拷贝文件夹下的内容进行 md5sum 验证



```
liuyin1813075@echo-virtual-machine:~/vmware-tools-distrib$ find ./ -type f -pri
nt0 | xargs -0 md5sum | sort -k 2 > /home/liuyin1813075/origin.txt
```



```
liuyin1813075@echo-virtual-machine:~/copyfile/tools_copy$ find ./ -type f -prin
t0 | xargs -0 md5sum | sort -k 2 > /home/liuyin1813075/newone.txt
```

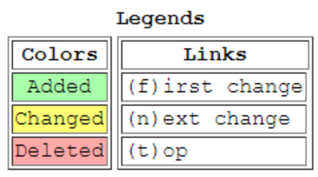
- 将生成的 origin.txt 和 newone.txt 文件导出到本地 windows 操作系统下
- 在 windows 下编写 python 代码比较 md5 值

Python 验证代码：

```python
import difflib
import sys


# 读取配置文件函数
def read_file(file_name):
    try:
        file_handle = open(file_name, 'r')
        text = file_handle.read().splitlines()  # 读取后以行进行分割
        file_handle.close()
        return text
    except IOError as error:
        print('Read file Error: {0}'.format(error))
        sys.exit()


# 比较两个文件并输出 html 格式的结果
```

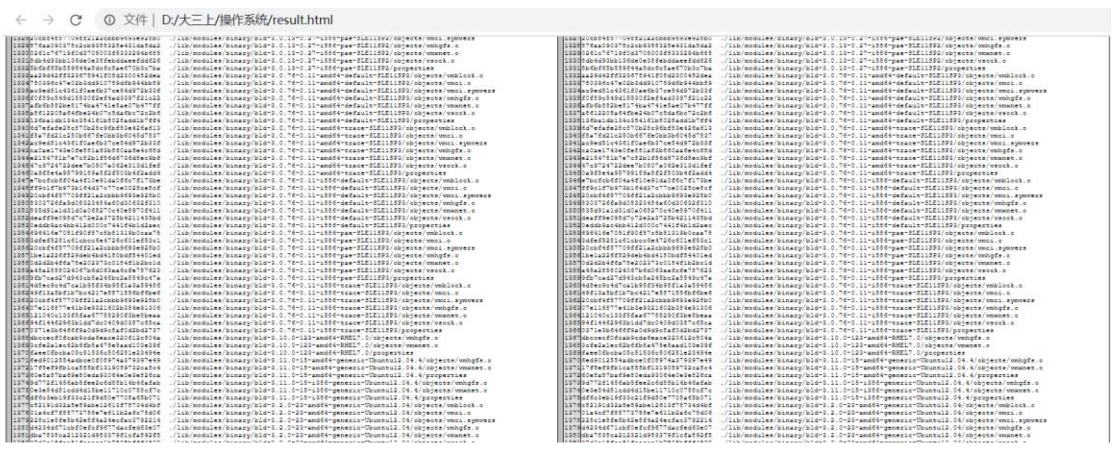
```
def compare_file(file1_name, file2_name):
    if file1_name == "" or file2_name == "":
        print('文件路径不能为空：file1_name 的路径为：{0}, file2_name 的路径为：{1} .'.format(file1_name, file2_name))
        sys.exit()
    text1_lines = read_file(file1_name)
    text2_lines = read_file(file2_name)
    diff = difflib.HtmlDiff()  # 创建 htmldiff 对象
    result = diff.make_file(text1_lines, text2_lines)  # 通过 make_file 方法输出 html 格式的对比结果
    #  将结果保存到 result.html 文件中并打开
    try:
        with open('result.html', 'w') as result_file:      #同 f = open('result.html', 'w') 打开或创建一个 result.html 文件
            result_file.write(result)                #同 f.write(result)
    except IOError as error:
        print('写入 html 文件错误：{0}'.format(error))


if __name__ == "__main__":
    compare_file('./newone.txt', './origin.txt')
```

- 在生成的 html 文件中进行查验。



Legends

Colors	Links
Added	(f)irst change
Changed	(n)ext change
Deleted	(t)op



发现没有颜色出现，全部对应相同。=>复制成功。