OS_Lab02_CopyDir

姓名	李懋
学号	2213189
邮箱	2213189@mail.nankai.edu.cn

1 实验题目

- 编写一个C/C++程序实现目录拷贝

2 实验目的

1. 编写C/C++程序
2. 实现拷贝一个目录以及它的所有子目录
3. 使用GCC编译
4. 使用IDE（集成开发环境）
5. 测试目录，从 www.kernel.org 下载最新的linux内核
    1. 下载压缩包
    2. 解压缩为linux-5.19.10文件夹
    3. 拷贝该文件夹到linux-5.19.10back文件夹
6. 验证文件夹副本是正确的

3 实验原理

1. 拷贝文件夹的原理方法：
    - 编写拷贝文件夹的函数readFileList，输入需要拷贝的文件夹路径sourcePath和目标文件夹路径targetPath
    - 首先判断源文件夹是否存在，若存在则循环读取其中的子文件/子文件夹，在源文件夹和目标文件夹的路径上加上子文件/子文件夹，通过ptr->d_type判断类型
        - 如果是文件夹，则在目标文件夹下创建文件夹并递归调用readFileList
        - 如果是链接文件，则调用link（source,target）
        - 如果是普通文件则调用自己写的copyfile函数，实现文件的拷贝
    - 并且使用了 clock()函数，统计程序运行的时间，与 Python 程序比较
2. Python版本思路与之相似
3. C具体代码：

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include <dirent.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>  // 引入 time.h 头文件

#define BUFSIZE 1024

// 拷贝文件的函数
int copyfile(const char* dest, const char* src) {
    FILE *fp1 = NULL, *fp2 = NULL;
    fp1 = fopen(src, "rb");
    if (fp1 == NULL) {
        perror("Open source file error");
        return -1;
    }
    fp2 = fopen(dest, "wb");
    if (fp2 == NULL) {
        perror("Open destination file error");
        fclose(fp1);
        return -2;
    }

    char buffer[BUFSIZE];
    size_t readLen, writeLen;
    while ((readLen = fread(buffer, 1, BUFSIZE, fp1)) > 0) {
        writeLen = fwrite(buffer, 1, readLen, fp2);
        if (readLen != writeLen) {
            perror("Write file error");
            fclose(fp1);
            fclose(fp2);
            return -3;
        }
    }

    fclose(fp1);
    fclose(fp2);
    return 0;
}

// 递归拷贝目录的函数
int readFileList(char* sourcePath, char* targetPath) {
    DIR *dir;
    struct dirent *ptr;
    char source[1000];
    char target[1000];

    // 打开源目录
    if ((dir = opendir(sourcePath)) == NULL) {
        perror("Open directory error");
        exit(1);
    }

    // 读取目录中的每个条目
```

```
    while ((ptr = readdir(dir)) != NULL) {
        // 跳过当前目录和父目录
        if (strcmp(ptr->d_name, ".") == 0 || strcmp(ptr->d_name, "..") == 0) {
            continue;
        }

        // 清空缓冲区
        memset(source, '\0', sizeof(source));
        memset(target, '\0', sizeof(target));

        // 构建源路径和目标路径
        strcpy(source, sourcePath);
        strcpy(target, targetPath);
        strcat(source, "/");
        strcat(target, "/");
        strcat(source, ptr->d_name);
        strcat(target, ptr->d_name);

        // 判断条目类型
        if (ptr->d_type == DT_DIR) { // 目录
            // 创建目标目录
            if (mkdir(target, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) != 0) {
                perror("Create directory error");
                exit(1);
            }
            // 递归拷贝子目录
            readFileList(source, target);
        } else if (ptr->d_type == DT_REG) { // 普通文件
            // 拷贝文件
            if (copyfile(target, source) != 0) {
                perror("Copy file error");
                exit(1);
            }
        } else if (ptr->d_type == DT_LNK) { // 链接文件
            // 拷贝链接
            if (symlink(ptr->d_name, target) != 0) {
                perror("Copy symlink error");
                exit(1);
            }
        }
    }

    closedir(dir);
    return 1;
}

int main(void) {
    char* sourcePath = "/home/limao2213189/Downloads/1/linux-6.10.10";
    char* targetPath = "/home/limao2213189/Downloads/2/linux-6.10.10-back";

    // 记录开始时间
    clock_t start_time = clock();

    // 创建目标目录
    if (mkdir(targetPath, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH) != 0) {
        perror("Create target directory error");
```

```
        exit(1);
    }

    // 开始拷贝
    readFileList(sourcePath, targetPath);

    // 记录结束时间
    clock_t end_time = clock();

    // 计算并输出运行时间
    double elapsed_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("Copy operation completed in %.2f seconds.\n", elapsed_time);

    return 0;
}
```

4. Python 具体代码:

```
import os
import shutil
import time

def copy_directory(source_path, target_path):
    # 如果目标目录不存在，则创建它
    if not os.path.exists(target_path):
        os.makedirs(target_path)

    # 遍历源目录中的所有文件和子目录
    for item in os.listdir(source_path):
        source_item = os.path.join(source_path, item)
        target_item = os.path.join(target_path, item)

        if os.path.isdir(source_item):
            # 如果是目录，递归调用copy_directory
            copy_directory(source_item, target_item)
        else:
            # 如果是文件，直接拷贝
            shutil.copy2(source_item, target_item)

def main():
    source_path = "/home/limao2213189/Downloads/1"
    target_path = "/home/limao2213189/Downloads/2"

    # 记录开始时间
    start_time = time.time()

    # 开始拷贝
    copy_directory(source_path, target_path)

    # 记录结束时间
    end_time = time.time()

    # 计算并输出运行时间
    elapsed_time = end_time - start_time
    print(f"Copy operation completed in {elapsed_time:.2f} seconds.")
```

```
if __name__ == "__main__":
    main()
```

4 实验具体步骤

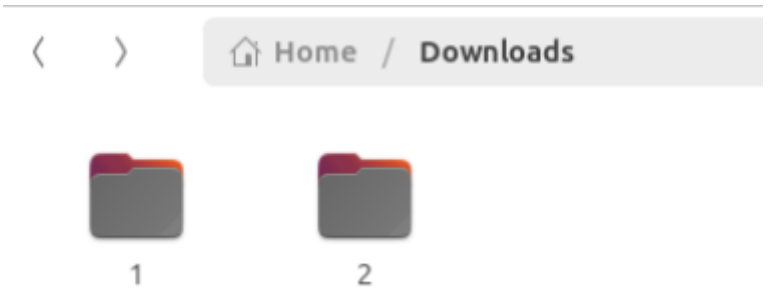
1. Install GCC(上次实验安装好了)
2. 安装 IDE，我这里选择更熟悉的vscode

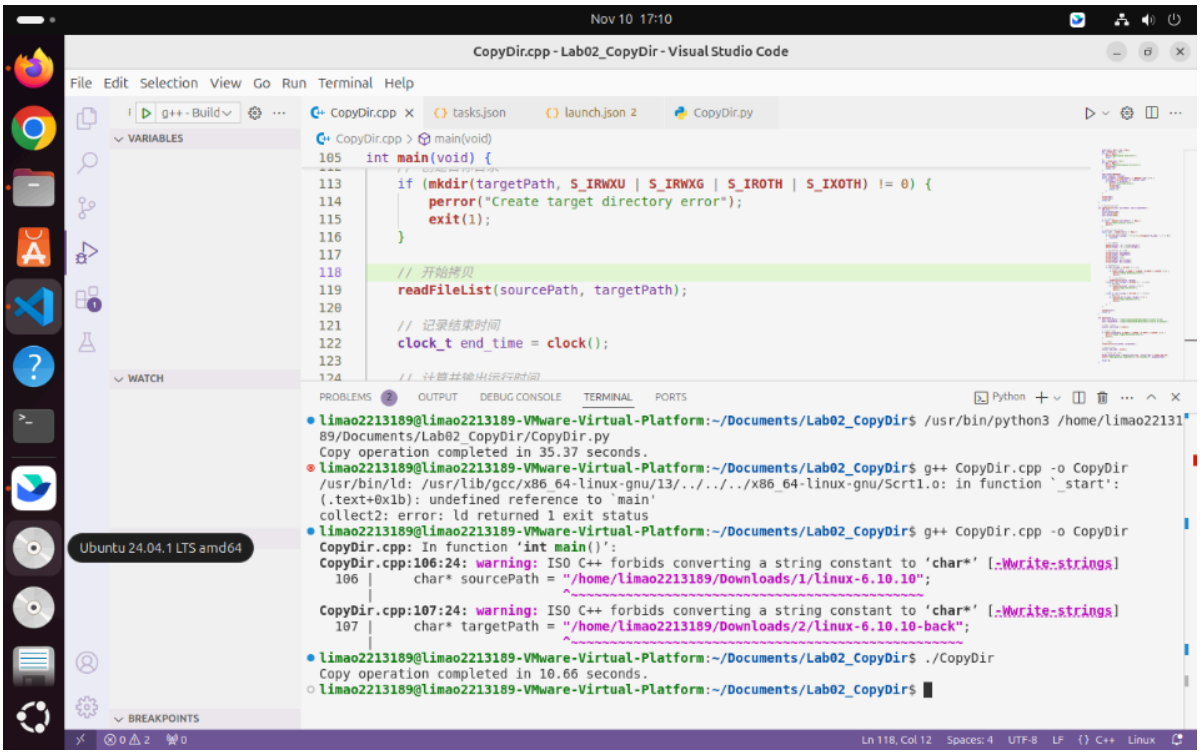
```
sudo snap install --class code
```

```
limao2213189@limao2213189-VMware-Virtual-Platform:~$ sudo snap install --classic code
Download snap "core20" (2434) from channel "stable"                    13%  145kB/s 6m40s
```

3. 下载并解压 Linux 内核

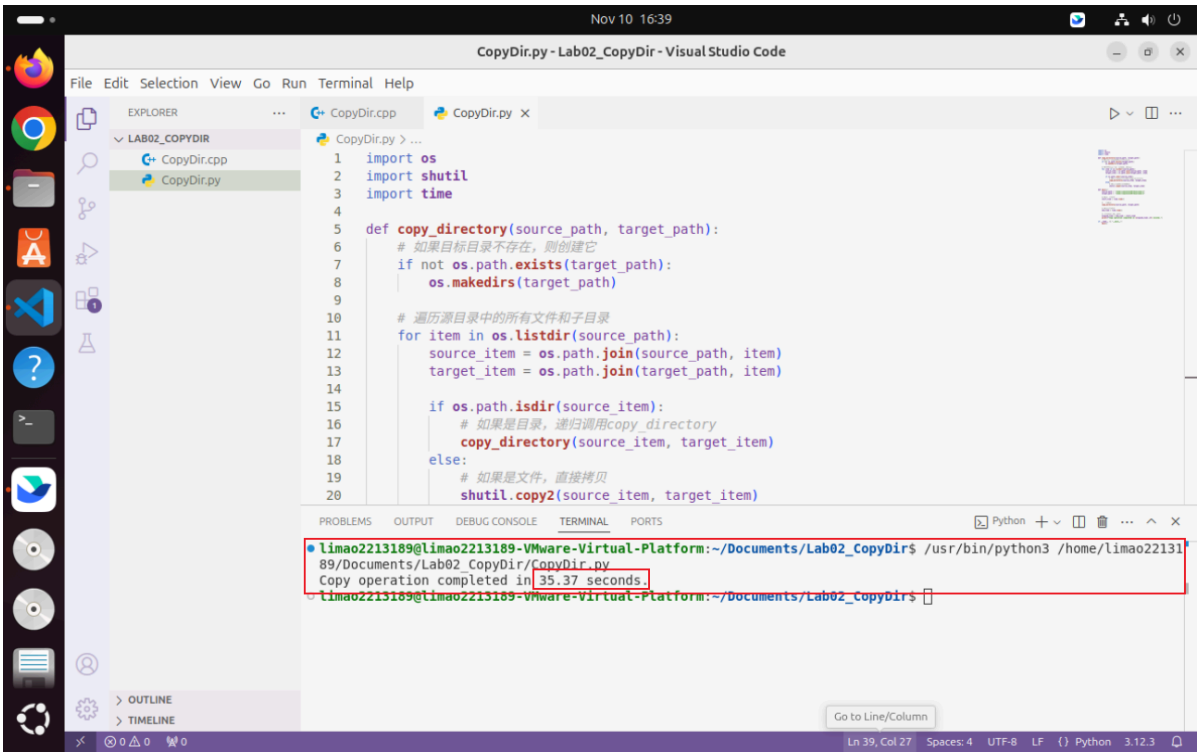
4. 使用 C 语言程序拷贝文件
   - 将 /home/limao2213189/Downloads/1 中的文件，拷贝到 /home/limao2213189/Downloads/2



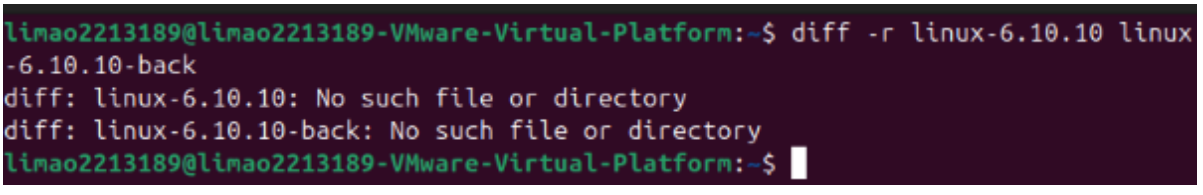


- **耗时 10.66s**

5. Python程序拷贝
    - **耗时 35.37s**



6. 验证正确性
    - diff指令没有找到不同的文件，说明两个文件夹完全相同，拷贝成功



5 实验总结

1. 拷贝一个文件夹及其所有文件夹主要运用的知识点是文件输入输出流以及递归思想。
2. 其中需要注意的是对于一个文件需要判断是文件夹还是链接文件还是常规文件，不同的类型对应不同的处理方式。
3. 验证两个文件夹是否相等的底层思路是MD5算法，利用哈希的思想，将数据映射为定长的编码。
4. C语言效率比Python高，得益于C是编译型语言，执行效率更高