**一．使用多线程拷贝代码**

这里使用2个线程来拷贝代码，一个线程负责将文件夹进行拷贝，另一个线程负责将具体的文件进行拷贝。代码如下：

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <unistd.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <sys/types.h>

#include <string.h>

#define MAX\_PATH\_LENGTH 1024

#define BUFFER\_SIZE 4096

**typedef** **struct** {

**char**\* source\_directory;

**char**\* destination\_directory;

} threadParm\_t;

**typedef** **struct** {

**char** src[512];

**char**\* dest[512];

**int** type;

} copy;

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

copy work\_queue[1001];

**int** queue\_count = 0;

**int** queue\_index = -1;

**char** destination\_directory[MAX\_PATH\_LENGTH];

**int** end\_of\_directory = 0;

**void** copy\_file(**const** **char** \*src, **const** **char** \*src2) {

**char** buffer[BUFFER\_SIZE];

**int** src\_fd = open(src, O\_RDONLY);

**int** src2\_fd = open(src2, O\_WRONLY | O\_CREAT, 0644);

**if** (src\_fd == -1 || src2\_fd == -1) {

        perror("文件打开失败");

        exit(1);

    }

    ssize\_t bytes\_read;

**while** ((bytes\_read = read(src\_fd, buffer, **sizeof**(buffer))) > 0) {

        write(src2\_fd, buffer, bytes\_read);

    }

    close(src\_fd);

    close(src2\_fd);

}

**void** copy\_directory(**const** **char** \* src, **const** **char** \* dest) {

    DIR \*dir = opendir(src);

**if** (dir == NULL) {

        perror("打开文件夹失败");

**return**;

    }

    mkdir(dest,0755);

**struct** dirent \*entry;

**struct** stat statbuf;

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

**char** src\_path[512];

**char** dest\_path[512];

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

**continue**;

        }

        snprintf(src\_path, **sizeof**(src\_path), "%s/%s", src, entry->d\_name);

        snprintf(dest\_path, **sizeof**(dest\_path), "%s/%s", dest, entry->d\_name);

**if** (lstat(src\_path, &statbuf) == -1) {

            perror("获取文件状态失败");

**continue**;

        }

**if** (S\_ISDIR(statbuf.st\_mode)) {

            copy\_directory(src\_path, dest\_path);

        }

        //if (S\_ISLNK(statbuf.st\_mode))

**else** {

            copy tem;

            strcpy(tem.src,src\_path);

            strcpy(tem.dest,dest\_path);

**if**(S\_ISLNK(statbuf.st\_mode))

            tem.type=1;

**else** tem.type=0;

**while**(queue\_index>=1000);

            pthread\_mutex\_lock(&mutex);

//            //忙等待

            queue\_index++;

            work\_queue[queue\_index] = tem;

            pthread\_mutex\_unlock(&mutex);

        }

    }

    closedir(dir);

}

**void** \*read\_directory(**void** \* parm) {

    threadParm\_t\* p = (threadParm\_t\*)parm;

**char**\* src = p->source\_directory;

**char**\* dest = p->destination\_directory;

    copy\_directory(src, dest);

    pthread\_mutex\_lock(&mutex);

    end\_of\_directory = 1;

    pthread\_cond\_broadcast(&cond);

    pthread\_mutex\_unlock(&mutex);

**return** NULL;

}

**void** copy\_symlink(**const** **char** \*src, **const** **char** \*dest) {

**char** target[512];

    ssize\_t len = readlink(src, target, **sizeof**(target) - 1);

**if** (len == -1) {

        perror("读取符号链接失败");

        exit(1);

    }

    target[len] = '\0'; // 确保字符串以 null 结尾

**if** (symlink(target, dest) == -1) {

        perror("创建符号链接失败");

        exit(1);

    }

}

**void** \*copy\_files(**void** \*arg) {

**while** (1) {

        copy a;

**while**(queue\_index==-1&&end\_of\_directory!=1);

**if**(end\_of\_directory==1&&queue\_index==-1)**break**;

        pthread\_mutex\_lock(&mutex);

        a = work\_queue[queue\_index];

        queue\_index--;

        pthread\_mutex\_unlock(&mutex);

**if**(a.type==0)

        copy\_file(a.src,a.dest);

**else** copy\_symlink(a.src,a.dest);

    }

**return** NULL;

}

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

**if** (argc < 3) {

        printf("Usage: %s <source\_directory> <destination\_directory>\n", argv[0]);

**return** 1;

    }

    pthread\_mutex\_init(&mutex, NULL);

    pthread\_cond\_init(&cond, NULL);

    threadParm\_t threadParm;

    threadParm.source\_directory= argv[1];

    threadParm.destination\_directory = argv[2];

    pthread\_t thread1;

    pthread\_t thread2;

    pthread\_create(&thread1, NULL, read\_directory, (**void**\*)&threadParm);

        pthread\_create(&thread2, NULL, copy\_files, (**void**\*)&threadParm);

        pthread\_join(thread1, NULL);

        pthread\_join(thread2, NULL);

    pthread\_mutex\_destroy(&mutex);

    pthread\_cond\_destroy(&cond);

**return** 0;

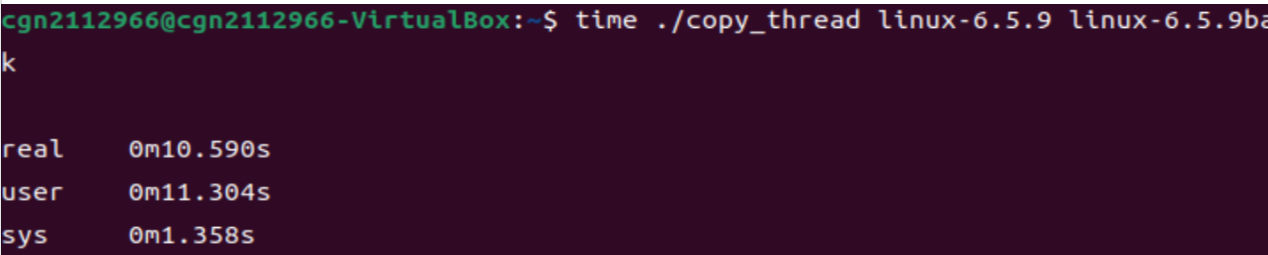
}

解析：创建一个queue来储存线程2要复制的文件的原始路径和目标路径(创建一个结构体来保存），queue为两个线程的临界区。缓冲区的大小设置为1000个文件地址。然后线程1执行read\_directory来进行文件夹的遍历，如果遍历到文件夹，则递归遍历，如果递归到文件，则进入临界区，将地址写入queue中并把queue\_index+1，表示可复制的文件数多了一个，当queue\_index>1000时，此时说明缓冲区已满，则需要忙等待。而线程2则执行copy\_files来进行文件的复制，当复制时需要读取缓冲区的地址，则需要上锁，如果queue\_index=-1，则说明此时缓冲区没有可以复制的文件，则需要忙等待。当文件夹遍历完且文件复制完时则跳出循环。

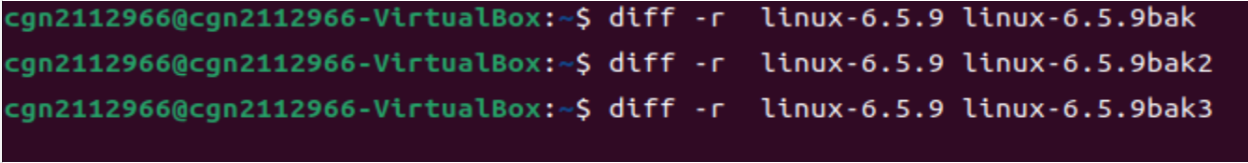
在执行copy\_file时需要看type是0还是1，如果是0则是普通文件，按普通文件进行复制，如果是1则是链接文件，按链接文件进行复制。

1. **执行后验证正确性**

先执行文件



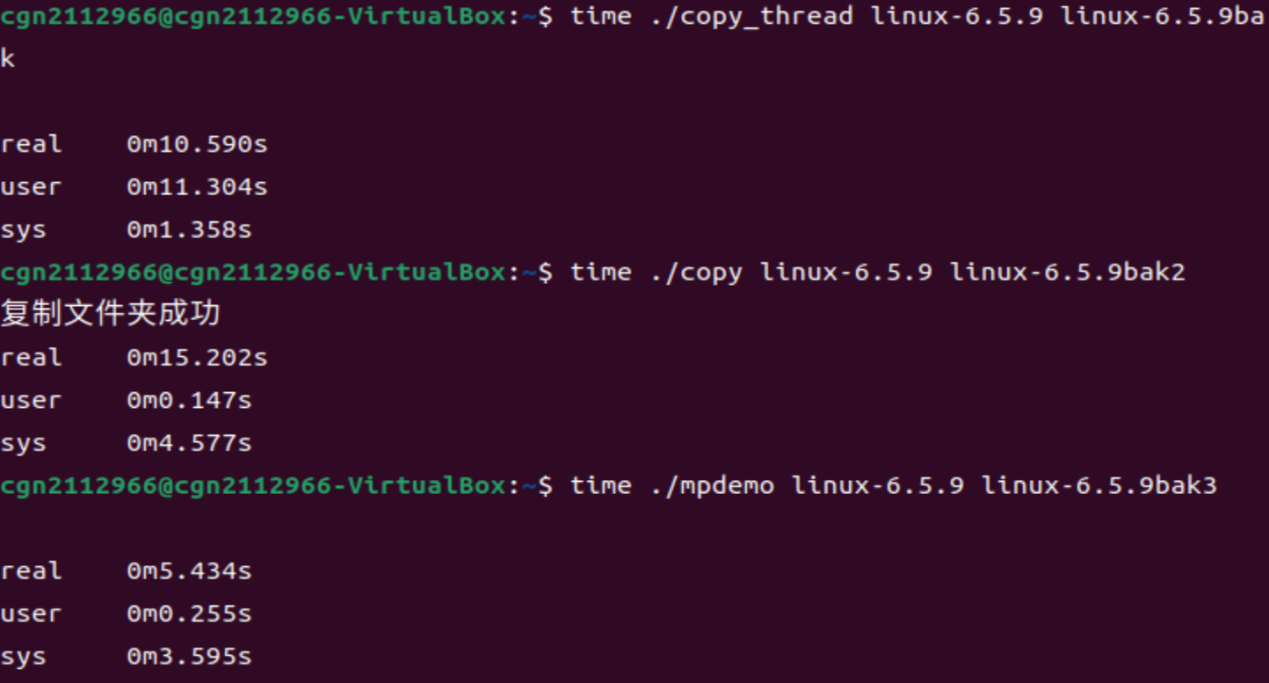
用diff -r 命令来验证正确性

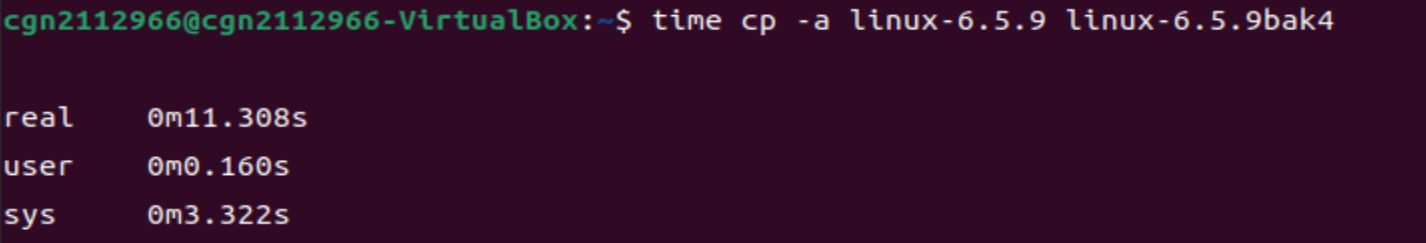


没有发现错误，说明复制成功。

1. **与多进程，单进程，cp命令的比较。**

先分别进行复制，用time 命令来看运行时间。





可以看出，多进程的运行时间<多线程的运行时间<系统cp命令的时间<单进程的运行时间。

因此拷贝效率多进程>多线程>系统cp命令>单进程。

1. **心得体会**

通过这次实验，我懂得了如何利用多线程来拷贝文件夹目录，并验证是否拷贝正确。我对临界区的互斥访问有了更深的理解，并了解了Pthread的用法。这让我受益匪浅。