**电子科技大学**信息与软件工程**学院**

**标 准 实 验 报 告**

**（实验）课程名称 操作系统原理与实践实验（二）**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：李金雨 学 号：2021090922015 指导教师：杨珊**

**实验地点：信软学院楼西304 实验时间： 2023-05-09**

# 实验项目名称：

1. 文件状态测试
2. 用户登录动态监测
3. 系统命令的实现

# 实验学时：

每个实验4学时，共12学时。

# 实验项目的目的和任务：

# 文件状态测试

目的：使学生熟悉UNIX的基本SHELL程序设计方法，包括命令行参数检测、变量设置、文件状态检测与特定信息读取、程序运行控制。

任务：编写一个SHELL程序，动态检测指定文件的状态信息，当文件的大小发生改变时，给出提示信息，并继续前进检测。当文件的大小的变化次数或持续检查无变化次数达到一定值时，退出检查，程序结束。

（1） 清屏；

（2） 提示用户输入要检测其状态的文件名；

（3） 显示该文件的状态信息（提示：该状态信息可由命令ls –l 来得到），或找不到该文件时的错误提示；

（4） 可用cut命令，或用sed或awk命令来截取状态信息中文件的大小并保存；

（5） 每隔5秒钟检测一次该文件大小的信息，并与保存的文件原来的大小相比较；

（6） 如果文件大小未改变，则屏幕显示不变，并继续每隔5秒钟检测一次；

（7） 如果文件大小已改变，则保存新的文件大小，并在屏幕上显示：

file [ filename ] size changed

方括号中的filename为本程序运行时用户输入的被检测的文件名。程序继续每隔5秒钟检测一次文件的大小；

（8） 程序循环执行5~7步的操作。当被检测的文件或者已累计改变了两次大小，或者已连续被检测了十次还未改变大小时，给出相应提示，然后清屏退出。

# 用户登录动态监测

目的：使学生熟悉一种简单的指定用户的监测方法. 练习命令行参数、用户变量、while循环控制和暂停进程（sleep）等方面的使用技巧。

任务：编写一个用户监测程序usr\_monitor, 其运行格式为：

usr\_monitor username

其中username是用户指定的任意一个用户名。程序运行时首先列出当前系统中的已登录用户的名单，再检查指定用户是否已登录。如果已登录，则显示相应信息；如果未登录，则等待该用户登录，直到指定用户登录进入系统为止。

（1） 建立shell程序usr\_monitor

（2） 根据变量$#的值检查命令行上是否有一个用户名，如果有则继续运行；如果没有或多于一个用户名则提示运行格式：

Usage: usr\_monitor username

（3） 运行who命令，并将运行结果中的用户名字段截取下来，保存到一个用户变量中。

（4） echo命令显示该变量的值，即当前系统中的用户名单。

（5） 判断用户名单中是否包含命令行上指定的用户，如果有则显示：

user [username] is logon

并退出程序。

（6） 如果用户名单中没有包含命令行上指定的用户，则显示：

waiting user [username] …

光标停留在省略号后面。

（7） 使用while循环，用sleep命令每隔5秒钟检查指定用户是否已登录系统。检测方法与步骤3相同。

（8） 如果指定用户未登录，则一直等待下去；如果指定用户已登录，则在省略号后面接着显示：

[username] is log on

退出循环，并结束程序。

# 系统命令的实现

目的：掌握Linux应用程序命令行参数传递方法；掌握POSIX API中文件I/O操作方法，包括：打开文件、关闭文件、创建文件、读写文件、定位文件。

任务：本实验主要是编程实现Linux下的cp -r命令。本实验主要是针对文件的I/O操作。本实验涉及的知识如下：

1、文件对象

文件对象是已打开的文件在内存中的表示，主要用于建立进程和磁盘上的文件的对应关系。文件对象和物理文件的关系类似进程和程序的关系，一个文件可能存在多个对应的文件对象。文件对象仅仅在进程观点上代表已经打开的文件，它反过来指向目录项对象。一个文件对应的索引节点和目录项对象是惟一的。

2、文件的打开、创建

①open函数

头文件：fcntl.h

int open( const char \*pathname, int oflag, …);

该函数打开或创建一个文件。其中第二个参数oflag说明打开文件的选项，第三个参数是变参，仅当创建新文件时才使用。

O\_RDONLY:：只读打开；O\_WRONLY：只写打开；O\_RDWR：读、写打开；O\_APPEND：每次写都加到文件尾；O\_CREAT：若此文件不存在则创建它，此时需要第三个参数mode，该参数约定了所创建文件的权限，计算方法为mode&~umask；O\_EXCL：如同时指定了O\_CREAT，此指令会检查文件是否存在，若不存在则建立此文件；若文件存在，此时将出错。O\_TRUNC：如果此文件存在，并以读写或只写打开，则文件长度0。由open返回的文件描述符一定是最小的未用描述符数值。

②creat函数

头文件：fcntl.h

int creat( const char \*pathname, mode\_t mode);

该函数用于创建一个新文件，其等效于open函数的如下调用：

open( pathname, O\_WROLY | O\_CREATE | O\_TRUNC, mode);

creat函数的一个不足之处是它以只写方式打开所创建的文件

早期UNIX版本中open的第二个参数只能是0、1或2，没有办法打开一个尚未存在的文件。如果要创建一个临时文件，并要先写该文件，然后又读该文件。则必须先调用creat，close，然后再open。

3、文件的关闭

头文件unistd.h

int close( int filedes );

该函数关闭以前打开的一个文件。关闭文件的同时也释放该进程加在该文件上的所有记录锁。当一个进程终止时，它所有的打开文件将由内核自动关闭。内核对文件描述符、对应的文件表项和索引节点表项进行相应的处理，来完成关闭文件的操作。进程关闭文件后，就不能通过该文件描述符操作该文件；当一个进程正常退出时，内核将关闭所有打开的文件描述符。

4、文件的读写

①read函数

头文件unistd.h

ssize\_t read( int filedes, void \*buf, size\_t nbytes);

read函数从打开的文件中读数据。如成功，则返回实际读到的字节数，如已到达文件的末尾或无数据可读，则返回0；有多种情况可使实际读到的字节数少于要求读的字节数：读普通文件，在读到要求字节数之前就到达文件尾；当从终端设备读，通常一次最多读一行；当从网络读时，网络中的缓冲机构可能造成返回值小于所要求读的字节数；某些面向记录的设备，如磁带，一次最多返回一个记录。读操作完成后，文件偏移量将从读之前的偏移量加上实际读的字节数。

②write函数

头文件unistd.h

ssize\_t write( int filedes, const void \*buf, size\_t nbytes);

该函数返回实际写的字节数，通常与参数nbytes的值相同，否则表示出错。如果出错，则返回－1。write出错的原因可能是磁盘满、没有访问权限、或写超过文件长度限制等等。对于普通文件，写操作从文件的当前偏移量开始写，除非打开文件时指定了O\_APPEND选项。完成后，文件偏移量加上实际写的字节数。

5、文件的定位

头文件unistd.h

每个打开的文件都有一个与其相关联的“当前文件偏移量”。当打开文件时，除指定O\_APPEND选项，文件偏移量默认为0。lseek函数用于改变文件的当前偏移量。

off\_t lseek( int filedes, off\_t offset, int whence);

对参数的解释与参数whence的值有关：若whence是SEEK\_SET,则将该文件的偏移量设置为距文件开始处的offset个字节；若whence是SEEK\_CUR,则将该文件的偏移量设置为距文件当前位置加offset个字节；若whence是SEEK\_END,则将该文件的偏移量设置为文件长度加offset个字节，offset可正可负；

6、目录操作-获取当前工作路径

常用函数：getcwd，get\_current\_dir\_name

头文件：unistd.h

函数定义：

char \*getcwd(char \*buf, size\_t size);将当前的工作目录绝对路径字符串复制到参数buf 所指的缓冲区，参数size为缓冲区大小 若参数buf 为NULL，参数size 为0，则函数根据路径字符串的长度自动分配缓冲区，并将分配的路径字符串缓冲区指针作为函数返回值（该内存区需要手动释放），失败返回NULL。

char \*get\_current\_dir\_name(void);成功返回路径字符串缓冲区指针（该内存区需要手动释放），失败返回NULL。

7、目录操作-打开关闭目录

常用函数：opendir，closedir

头文件：dirent.h

函数定义：

DIR \* opendir(const char \* name);打开参数name指定的目录，并使一个目录流与它关联，目录流类似于C库函数中的文件流，失败返回NULL。

int closedir(DIR \*dir);关闭指定目录流，释放相关数据结构，成功返回0；失败返回-1。

8、目录操作-读取目录文件

常用函数：readdir

头文件：sys/types.h；dirent.h

函数定义：

struct dirent \* readdir(DIR \* dir);读取目录流标识的目录文件，目录文件是一系列目录项的列表，每执行一次readdir，该函数返回指向当前读取目录项的指针，如果到达目录结尾或者有错误发生则返回NULL。

9、目录操作-读取目录文件

重要数据结构

struct dirent  
{  
 ino\_t d\_ino; i节点号  
 off\_t d\_off; 在目录文件中的偏移  
 usigned short d\_reclen; 文件名长度  
 unsigned char d\_type; 文件类型  
 char d\_name[256];文件名  
};

10、文件属性管理-读取文件属性

常用函数：stat，lstat，fstat

头文件： sys/stat.h

函数定义：

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

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

两个函数参数相同，功能类似，读取path参数所指定文件的文件属性并将其填充到buf参数所指向的结构体中，对于符号链接文件，lstat返回符号链接的文件属性，stat返回符号链接引用文件的文件属性。

int fstat(int filedes, struct stat \*buf);

与前两个函数功能类似，指定文件的方式改为通过文件描述符。

11、文件属性管理-文件属性解析

重要数据结构

struct stat {

mode\_t st\_mode; 文件类型与访问权限

ino\_t st\_ino; i节点号

dev\_t st\_dev; 文件使用的设备号

dev\_t st\_rdev; 设备文件的设备号

nlink\_t st\_nlink; 文件的硬链接数

uid\_t st\_uid; 文件所有者用户ID

gid\_t st\_gid; 文件所有者组ID

off\_t st\_size; 文件大小（以字节为单位）

time\_t st\_atime; 最后一次访问该文件的时间

time\_t st\_mtime; 最后一次修改该文件的时间

time\_t st\_ctime; 最后一次改变该文件状态的时间

blksize\_t st\_blksize; 包含该文件的磁盘块的大小

blkcnt\_t st\_blocks; 该文件所占的磁盘块 数

};

12、判定文件类型常用宏

UNIX或Linux系统中的常见文件类型有：

普通文件: S\_ISREG()

目录文件 S\_ISDIR()

字符特殊文件 S\_ISCHR()

块特殊文件 S\_ISBLK()

FIFO文件 S\_ISFIFO()

套接口文件 S\_ISSOCK()

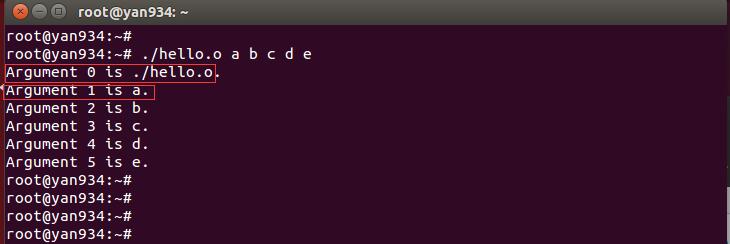
符号连接 S\_ISLINK()

13、在Linux程序中main函数定义：

int main(int argc, char\*\* argv)

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

参数意义：argc 是指命令行输入参数的个数；argv存储了命令行参数（至少一个）



其中Argument 0是可执行文件名本身，Argument 1开始才是真正的命令行参数，以字符串的形式传递（空格作为命令行参数之间的分隔）。

# 实验内容：

# 文件状态测试

编写一个SHELL程序，动态检测指定文件的状态信息，当文件的大小发生改变时，给出提示信息，并继续前进检测。当文件的大小的变化次数或持续检查无变化次数达到一定值时，退出检查，程序结束。

shell程序设计思路：

1. 清屏；
2. 提示用户输入要检测其状态的文件名；
3. 显示该文件的状态信息（提示：该状态信息可由命令ls –l 来得到），或找不到该文件时的错误提示；
4. 用cut命令，或用sed或awk命令来截取状态信息中文件的大小并保存；
5. 每隔5秒钟检测一次该文件大小的信息，并与保存的文件原来的大小相比较；
6. 如果文件大小未改变，则屏幕显示不变，并继续每隔5秒钟检测一次；
7. 如果文件大小已改变，则保存新的文件大小，并在屏幕上显示：file [ filename ] size changed（括号中的filename为本程序运行时用户输入的被检测的文件名）。程序继续每隔5秒钟检测一次文件的大小；
8. 程序循环执行5~7步的操作。当被检测的文件或者已累计改变了两次大小，或者已连续被检测了十次还未改变大小时，给出相应提示，然后清屏退出。

# 用户登录动态监测

编写一个用户监测程序usr\_monitor, 其运行格式为：usr\_monitor username

其中username是用户指定的任意一个用户名。程序运行时首先列出当前系统中的已登录用户的名单，再检查指定用户是否已登录。如果已登录，则显示相应信息；如果未登录，则等待该用户登录，直到指定用户登录进入系统为止。

1. 建立shell程序usr\_monitor
2. 根据变量$#的值检查命令行上是否有一个用户名，如果有则继续运行；如果没有或多于一个用户名则输出提示信息：Usage: usr\_monitor username
3. 运行who命令，并将运行结果中的用户名字段截取下来，保存到一个用户变量中。
4. echo命令显示该变量的值，即当前系统中的用户名单。
5. 判断用户名单中是否包含命令行上指定的用户，如果有则输出提示信息：user [username] is logon，并退出程序。
6. 如果用户名单中没有包含命令行上指定的用户，则输出提示信息：waiting user [username] ...，光标停留在省略号后面。
7. 使用while循环，用sleep命令每隔5秒钟检查指定用户是否已登录系统。检测方法与步骤2相同。
8. 如果指定用户未登录，则一直等待下去；如果指定用户已登录，则在省略号后面接着输出提示信息： [username] is log on，退出循环，并结束程序。

# 系统命令的实现

编写Linux应用程序，利用POSIX API在Linux系统上编写应用程序，实现cp命令的部分功能，主要实现文件到文件的复制功能（执行“cp 源文件路径 目标文件路径”实现的功能）。

在实现基本功能实现的基础上，针对各种特殊情况和边界条件等进行流程的完善与优化，包括：

a)目标文件存在时给出“是否覆盖”的提示信息,用户可以选择覆盖目标文件或者将已经存在的目标文件和源文件实现合并（在目标文件的尾部实现追加写入）

b)源文件不存在时给出出错提示信息

c)源文件是目录文件时给出错误提示信息

# 实验器材（设备、元器件）：

PC计算机，操作系统：Ubuntu

# 实验步骤：

1. 搭建开发环境

只需要安装好Ubuntu系统即可，这里使用了Windows的Linux子系统（Ubuntu内核）。

Ubuntu环境的安装：

* 1. 启动Powershell
  2. 执行wsl –install命令
  3. 如图，安装完成后执行wsl命令即可进入Ubuntu系统命令行

文本

描述已自动生成

1. 程序开发：
   1. 文件状态测试

流程图如图：

图表, 图示

描述已自动生成

* 1. 用户登录动态监测

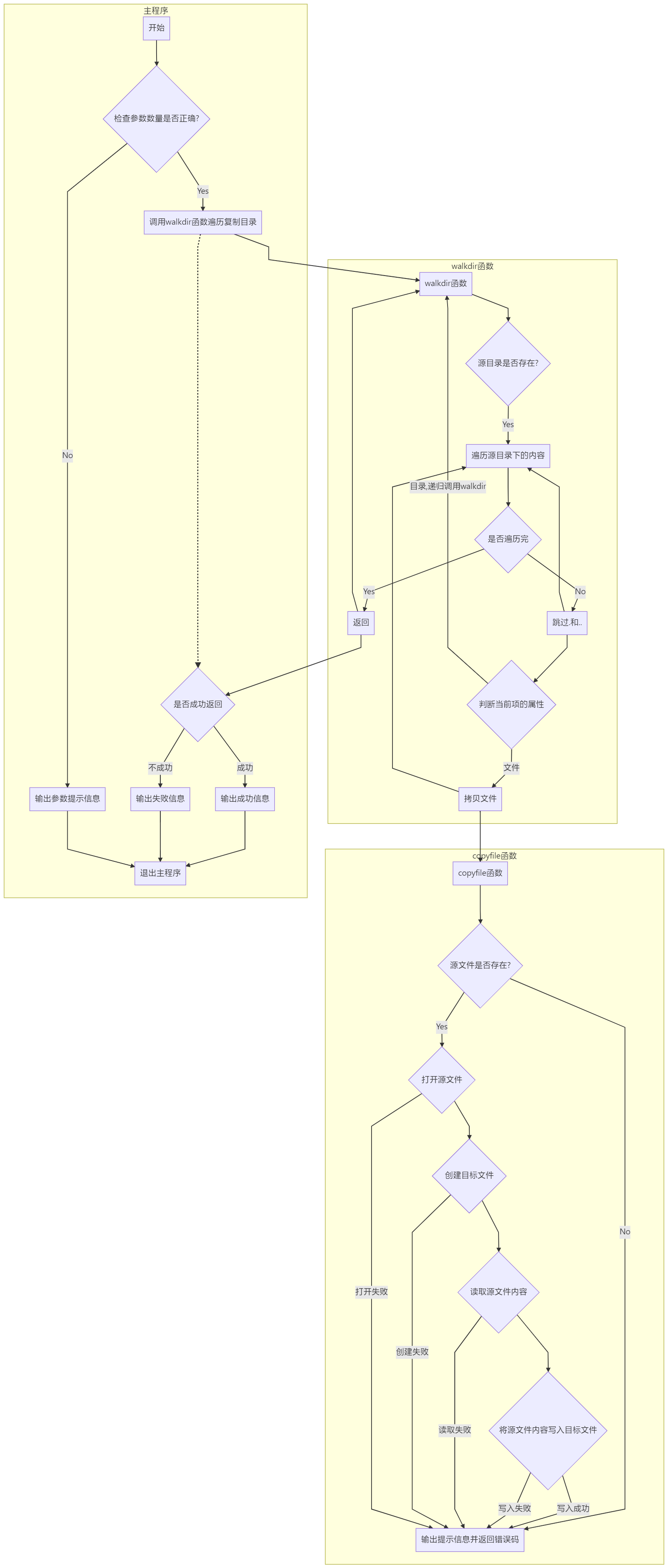
流程图如图：

图示

描述已自动生成

* 1. 系统命令的实现

流程图如图：



3) 编译、调试、运行

* 命令行下的编译运行
  + C语言的编译：使用gcc命令（对于使用了线程的，需要加上-lpthread选项），如对lab1文件，执行gcc lab1.c -o lab1 -lpthread，可以看到生成了lab1可执行文件：

文本

描述已自动生成

* 可执行文件（包括C语言编译生成的文件和sh文件）的执行：如对lab1文件，执行./lab1即可：

文本

中度可信度描述已自动生成

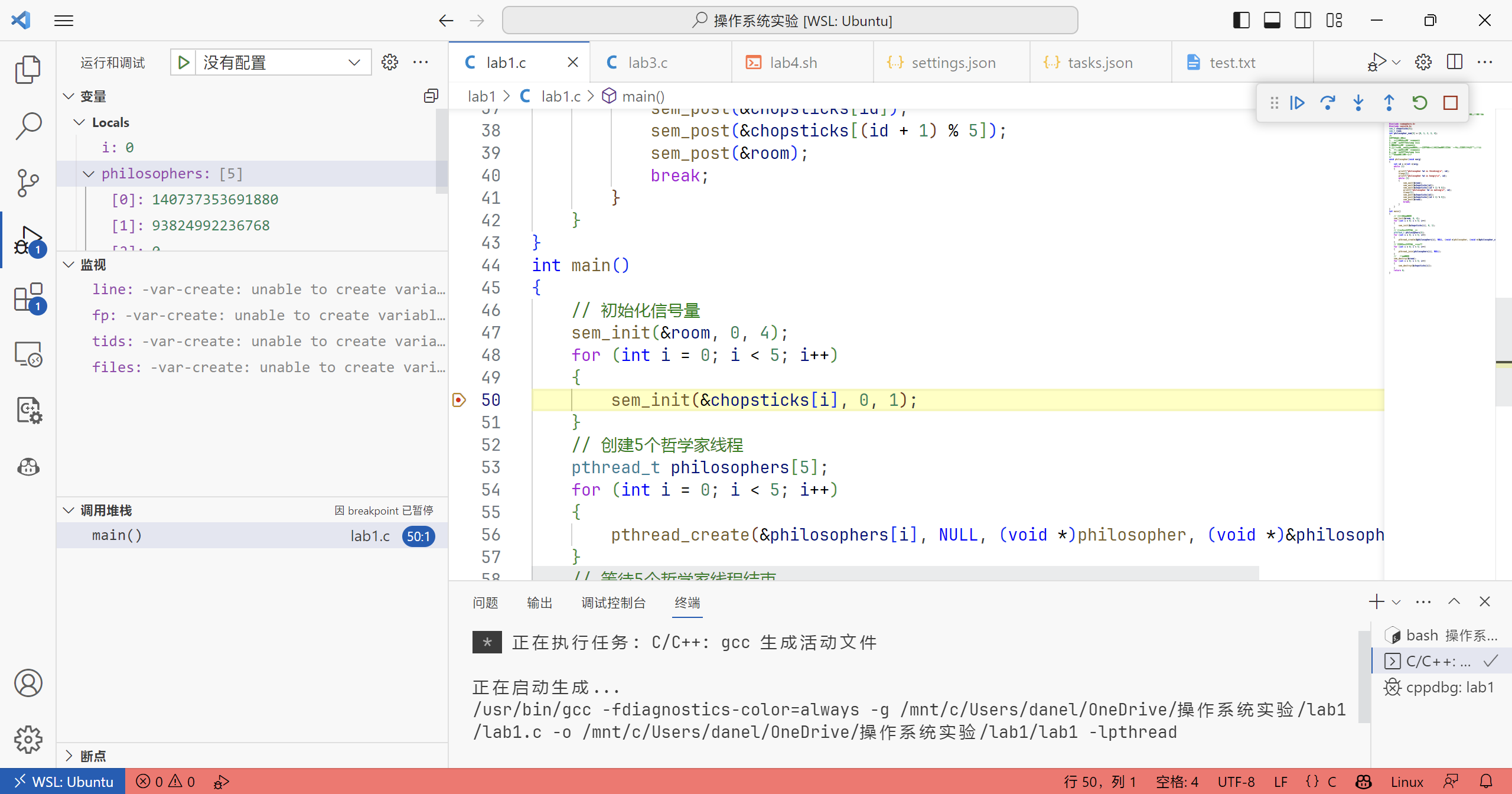
* VSCode下的编译、调试、运行
  + C语言的运行：直接点击运行按钮即可，由于VSCode可以直接支持使用 WSL 进行远程开发和调试，点击运行按钮后会自动在Ubuntu系统下编译和运行程序：

文本

低可信度描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

* + C语言的调试：直接点击调试按钮即可，由于VSCode可以直接支持使用 WSL 进行远程开发和调试，点击运行按钮后会自动在Ubuntu系统下编译和调试程序（下图设置了一个断点）：
  + 

# 实验数据及结果分析：

# 文件状态测试

程序代码如下：

|  |
| --- |
| clear  echo "Please input the filename to be detected:"  read filename  if [ ! -e $filename ]; then      echo "File ${filename} not found!"      exit 1  fi  echo "Current status of ${filename}:"  ls -l $filename  filesize=`ls -l $filename | cut -d " " -f 5`  unchanged=0  changed=0  while true; do      sleep 5      if [ $filesize -eq `ls -l $filename | cut -d " " -f 5` ]; then          unchanged=`expr $unchanged + 1`          echo "File ${filename} size unchanged."          if [ $unchanged -eq 10 ]; then              echo "The file has not been changed for 10 times."              break          fi      else          filesize=`ls -l $filename | cut -d " " -f 5`          echo "File ${filename} size changed."          unchanged=0          changed=`expr $changed + 1`          if [ $changed -eq 2 ]; then              echo "The file has been changed for 2 times."              break          fi      fi  done |

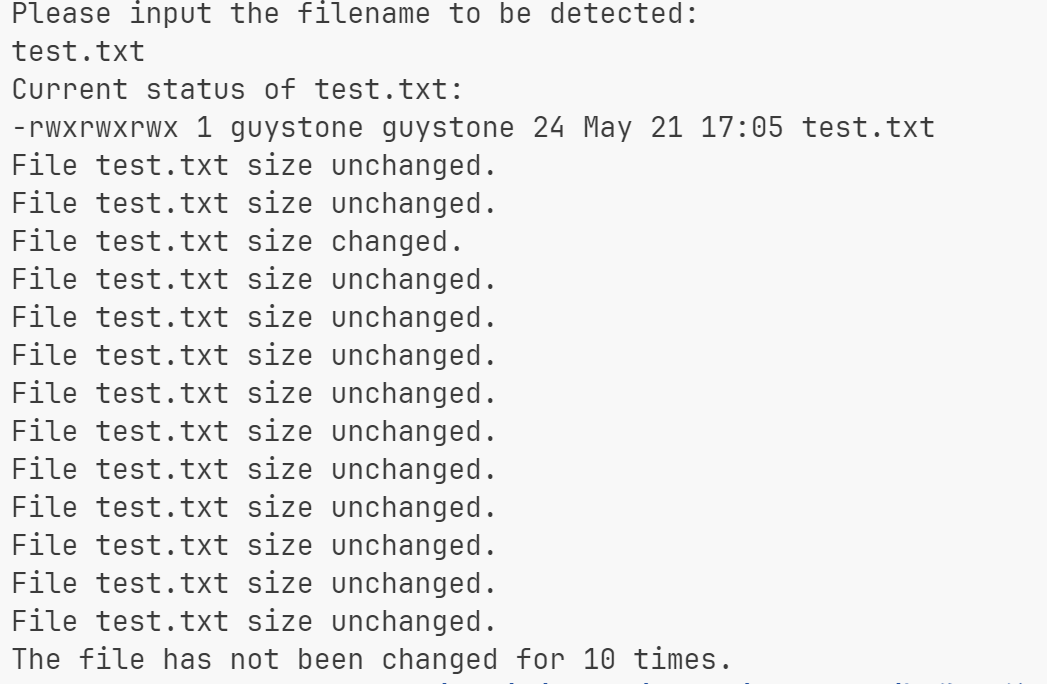
执行结果：

累计改变两次的情况：

文本

描述已自动生成

连续十次未改变大小的情况：



# 用户登录动态监测

程序代码如下：

|  |
| --- |
| if [ $# -ne 1 ]; then      echo "Usage: usr\_monitor username"      exit 1  fi  username=$1  while true; do      users=(`who | cut -d " " -f 1`) # 使用who列出当前登录的用户并截取用户名存入数组（括号是为了转化为数组）      echo "Current user list:${users[@]}"      for user in ${users[@]}; do          if [ $user == $username ]; then              echo "user ${username} is logon"              exit 0          fi      done      echo "waiting user ${username} ..."      sleep 5  done |

执行结果：

可以看到在右边的终端中登入root用户后左边终端中运行的程序检测到root用户登入。

（注：使用的是Windows的Linux子系统（Ubuntu内核），wsl sudo login -f root命令的意思是在Ubuntu中登录root用户）

文本

描述已自动生成

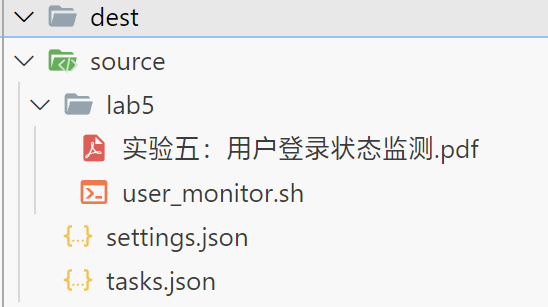
# 系统命令的实现

程序代码如下：

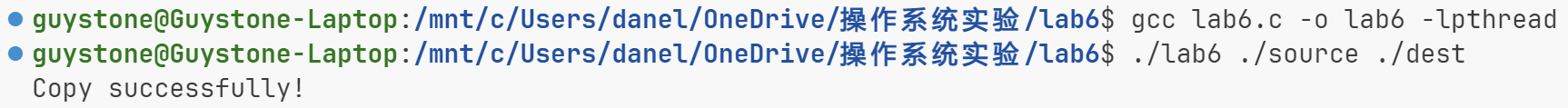
|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <dirent.h>  #include <fcntl.h>  #include <string.h>  #include <sys/stat.h>  #include <sys/time.h>  // 首先实现文件到文件的拷贝  int copyfile(char \*spath, char \*dpath)  {      int sfd, dfd;      if ((sfd = open(spath, O\_RDONLY)) == -1) // 只读      {          printf("Source file does not exist!\n");          return -1;      }      if (dfd = open(dpath, O\_WRONLY) == -1) // 目标文件不存在则创建      {          if ((dfd = open(dpath, O\_WRONLY | O\_CREAT, 0644)) == -1) // 只写，不存在则创建          {              printf("Destination file create error!\n");              return -1;          }      }      else      {          printf("File %s already exists, overwrite it(1) or skip it(else)?\n", dpath);          char choice;          while ((choice = getchar()) == '\n')              ;          if (choice == '1')          {              if ((dfd = open(dpath, O\_WRONLY | O\_TRUNC, 0644)) == -1) // 覆盖文件              {                  printf("Destination file create error!\n");                  return -1;              }          }          else              return 0; // 跳过      }      char buf[1024]; // 缓冲区      int n;      while ((n = read(sfd, buf, 1024)) > 0) // 读取源文件,到达文件尾部时返回0      {          if (n == -1 || (write(dfd, buf, n)) == -1) // 写入目标文件          {              printf("Error copying file from %s to %s\n", spath, dpath);              return -1;          }      }      close(sfd);      close(dfd);      return 0;  }  // 实现文件属性的判断  int isdir(char \*path)  {      struct stat buf;      if (stat(path, &buf) == -1)      {          printf("Error getting file status!\n");          return -1;      }      return S\_ISDIR(buf.st\_mode);  }  // 然后实现目录的遍历及复制  int walkdir(char \*spath, char \*dpath)  {      DIR \*sdir;      struct dirent \*sdp;      if ((sdir = opendir(spath)) == NULL)      {          printf("Source directory does not exist!\n");          return -1;      }      while ((sdp = readdir(sdir)) != NULL)      {          if (strcmp(sdp->d\_name, ".") == 0 || strcmp(sdp->d\_name, "..") == 0) // 忽略.和..              continue;          char sfile[1024], dfile[1024];          // 拼接源文件和目标文件的路径          strcpy(sfile, spath);          strcat(sfile, "/");          strcat(sfile, sdp->d\_name);          strcpy(dfile, dpath);          strcat(dfile, "/");          strcat(dfile, sdp->d\_name);          if (isdir(sfile))          {              // 如果目录不存在则创建              DIR \*temp = opendir(dfile);              if (temp != NULL)                  closedir(temp);              else if (mkdir(dfile, 0777) == -1) // 创建目录              {                  printf("Error creating directory %s\n", dfile);                  return -1;              }              if (walkdir(sfile, dfile) == -1) // 递归遍历子目录                  return -1;          }          else          {              if (copyfile(sfile, dfile) == -1) // 拷贝文件                  return -1;          }      }      closedir(sdir);      return 0;  }  // 主函数实现参数读取  int main(int argc, char \*argv[])  {        if (argc != 3)      {          printf("Usage: %s source\_path destination\_path\n", argv[0]);          return -1;      }      if ((walkdir(argv[1], argv[2])) == 0)          printf("Copy successfully!\n");      else      {          printf("Copy failed!\n");          return -1;      }      return 0;  } |

执行结果：

拷贝前：



执行：



执行后：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

# 总结及心得体会：

通过这次实验,我加深了对Shell脚本编程和Linux系统API的了解。

在文件状态测试部分,我熟悉了Shell脚本中的变量定义、条件判断、循环结构等基本语法,掌握了ls、cut等命令来获取和解析文件信息。通过周期性读取文件状态并做比较,我实现了程序动态监控文件的变化。

在用户登录监测部分,我熟悉了Shell程序获取和解析命令行参数的方法,利用who命令获取当前用户列表并截取用户名,通过while循环和sleep实现周期性读取和判断。这为我日后编写Shell脚本程序打下了基础。

在系统命令实现部分,我熟练掌握了Linux系统中文件和目录的操作API,包括打开、关闭、创建、移动、复制等功能。我通过打开文件读取源文件内容并写入目标文件实现了文件拷贝,通过递归遍历目录结构实现了目录拷贝。同时,我对于C语言在Linux下的编译运行调试更加熟练了。

总的来说,本次实验加深了我对Linux系统编程的理解,掌握了Shell脚本和C语言在Linux下的使用方法。这为我以后的学习和工作奠定了基础。同时,通过实际操作,我对Linux文件系统的概念结构也更加直观了。

# 对本实验过程及方法、手段的改进建议：

1. 实验内容可以更加丰富，比如:

* 用户登录监测可以增加更多功能,比如记录用户登录次数、登录成功与失败记录等。
* 文件状态检测可以增加更多条件,比如文件被删除时也给出提示。
* 系统命令实现可以增加更多验证条件,比如源路径和目标路径不能相同等。

1. 对于文件拷贝，可以使用多线程/多进程来实现，以加快拷贝速度

下列代码是一个基于本实验代码加入多线程的示例，通过测试可以达到减少70%拷贝时间的效果（基于原实验代码测试）：

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <dirent.h>  #include <fcntl.h>  #include <string.h>  #include <sys/stat.h>  #include <sys/time.h>  #include <pthread.h>  #include <stdlib.h>  struct file\_paras  {      char \*spath;      char \*dpath;  };  int files = 0;  struct file\_paras \*fileQueue[1024]; // 文件队列  void appendFileQueue(struct file\_paras \*fp)  {      fileQueue[files++] = fp;  }  // 首先实现文件到文件的拷贝  void copyfile(void \*arg)  {      struct file\_paras \*fp = (struct file\_paras \*)arg;      char \*spath = fp->spath;      char \*dpath = fp->dpath;      int sfd, dfd;      if ((sfd = open(spath, O\_RDONLY)) == -1) // 只读      {          printf("Source file does not exist!\n");          exit(-1);      }      // 目标文件不存在则创建，存在则覆盖      if ((dfd = open(dpath, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644)) == -1) // 只写，不存在则创建      {          printf("Destination file create error!\n");          exit(-1);      }      char buf[1024]; // 缓冲区      int n;      while ((n = read(sfd, buf, 1024)) > 0) // 读取源文件,到达文件尾部时返回0      {          if (n == -1 || (write(dfd, buf, n)) == -1) // 写入目标文件          {              printf("Error copying file from %s to %s\n", spath, dpath);              exit(-1);          }      }      printf("%s copied to %s\n", spath, dpath);      close(sfd);      close(dfd);      return;  }  // 实现文件属性的判断  int isdir(char \*path)  {      struct stat buf;      if (stat(path, &buf) == -1)      {          printf("Error getting file status!\n");          exit(-1);      }      return S\_ISDIR(buf.st\_mode);  }  // 然后实现目录的遍历及复制  int walkdir(char \*spath, char \*dpath)  {      DIR \*sdir;      struct dirent \*sdp;      if ((sdir = opendir(spath)) == NULL)      {          printf("Source directory does not exist!\n");          exit(-1);      }      while ((sdp = readdir(sdir)) != NULL)      {          if (strcmp(sdp->d\_name, ".") == 0 || strcmp(sdp->d\_name, "..") == 0) // 忽略.和..              continue;          char sfile[1024], dfile[1024];          // 拼接源文件和目标文件的路径          strcpy(sfile, spath);          strcat(sfile, "/");          strcat(sfile, sdp->d\_name);          strcpy(dfile, dpath);          strcat(dfile, "/");          strcat(dfile, sdp->d\_name);          if (isdir(sfile))          {              // 如果目录不存在则创建              DIR \*temp = opendir(dfile);              if (temp != NULL)                  closedir(temp);              else if (mkdir(dfile, 0777) == -1) // 创建目录              {                  printf("Error creating directory %s\n", dfile);                  exit(-1);              }              if (walkdir(sfile, dfile) == -1) // 递归遍历子目录                  exit(-1);          }          else          {              // 如果是文件则直接加入文件队列等待拷贝              struct file\_paras \*fp = (struct file\_paras \*)malloc(sizeof(struct file\_paras));              char \*sp = (char \*)malloc(sizeof(char) \* 1024);              char \*dp = (char \*)malloc(sizeof(char) \* 1024);              strcpy(sp, sfile);              strcpy(dp, dfile);              fp->spath = sp;              fp->dpath = dp;              appendFileQueue(fp);          }      }      closedir(sdir);      return 0;  }  // 主函数实现参数读取  int main(int argc, char \*argv[])  {      if (argc != 3)      {          printf("Usage: %s source\_path destination\_path\n", argv[0]);          exit(-1);      }      // 计算拷贝时间      struct timeval start, end;      gettimeofday(&start, NULL);      if ((walkdir(argv[1], argv[2])) != 0)      {          printf("Copy failed!\n");          exit(-1);      }      // 多线程拷贝      pthread\_t tid[files];      for (int i = 0; i < files; i++)      {          pthread\_create(&tid[i], NULL, (void \*)copyfile, (void \*)fileQueue[i]);      }      // 等待线程结束      for (int i = 0; i < files; i++)      {          pthread\_join(tid[i], NULL);      }      gettimeofday(&end, NULL);      long timeuse = 1000000 \* (end.tv\_sec - start.tv\_sec) + end.tv\_usec - start.tv\_usec;      printf("time = %ld us\n", timeuse);      return 0;  } |

1. 对于系统命令实现这个实验，实验内容可以拆分成多个,一个实验集中一个功能的实现，这样代码看起来不会那么臃肿。

**报告评分：**

**指导教师签字：**