**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**

**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： 高豪 学 号：202090915005 指导教师：任立勇**

**实验地点： 信软楼西305 实验时间： 2022.5**

1. **实验室名称：**

信软楼西305

1. **实验项目名称：**

文件状态测试

1. **实验学时：**

4学时

1. **实验原理：**

SHELL编程原理

1. **实验目的：**

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。

1. **实验内容：**

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. **实验器材（设备、元器件）：**

PC计算机，操作系统：Ubuntu

1. **实验步骤：**

（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；

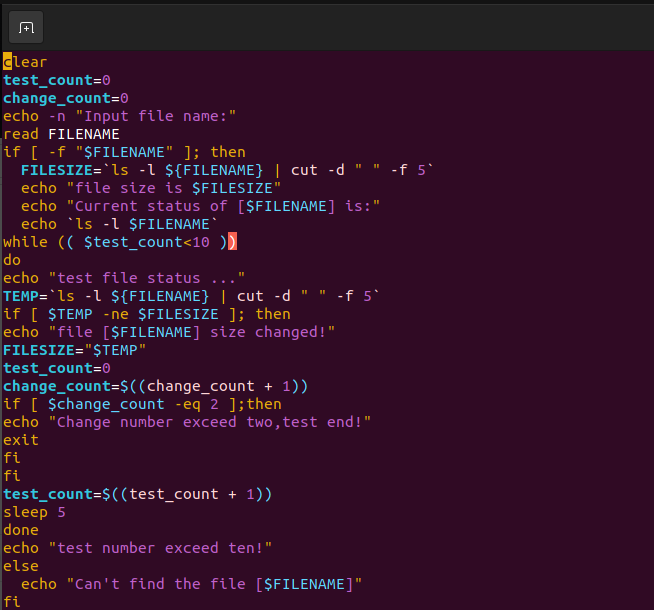
（7）如果文件大小已改变，则保存新的文件大小，并在屏幕上显示：

file [ filename ] size changed

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；

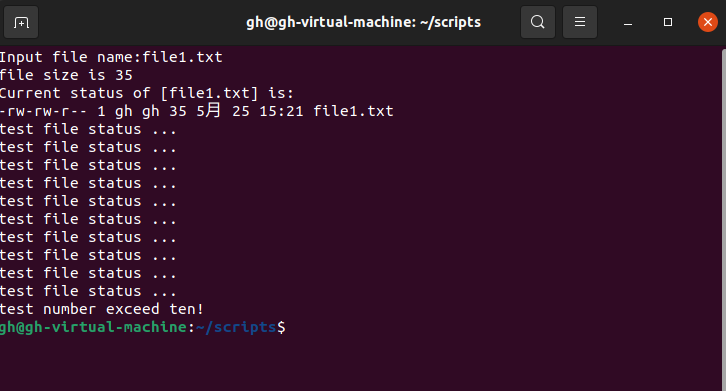
（8）程序循环执行5~7步的操作。当被检测的文件或者已累计改变了两次大小，或者已连续被检测了十次还未改变大小时，给出相应提示，然后清屏退出。

代码如下：



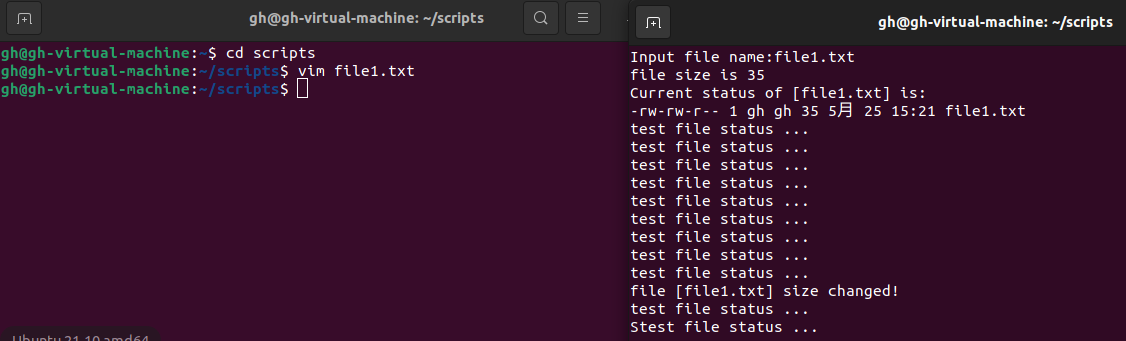
1. **实验数据及结果分析：**

1. 运行程序，输入要检查的文件名，在十次检查周期内，文件大小都一直不变的情况下，屏幕显示如下：

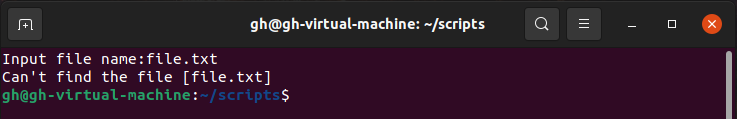


1. 运行程序后，被检查的文件的大小改变了两次时，屏幕显示的运行情况：

在左侧测试运行的过程中，我打开了需要修改的文件，在进行了修改之后，会发现右侧检测时显示了文件的大小被改变了。



1. 当文件夹中没有对应文件时：



1. **总结及心得体会：**

**明白了如何创建一个shell脚本，熟悉了整个创建的步骤：**

**①cat > experiment.sh**

**②chmod u+x experiment.sh**

**③bash experiment.sh**

**同时熟悉了在shell脚本程序下如何创建文件并进行文件大小状态的检测**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名： 高豪 学 号：2020090915005 指导教师：任立勇**

**实验地点： 信软楼西305 实验时间： 2022.5**

1. **实验室名称：**

信软楼西305

1. **实验项目名称：**

用户登录动态监测

1. **实验学时：**

4学时

**四、实验原理：**

SHELL编程原理

**五、实验目的：**

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。

**六、实验内容：**

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。

**七、实验器材（设备、元器件）：**

PC计算机，操作系统：Ubuntu

**八、实验步骤：**

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

1. 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。
2. echo命令显示该变量的值，即当前系统中的用户名单。
3. 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。

1. 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。

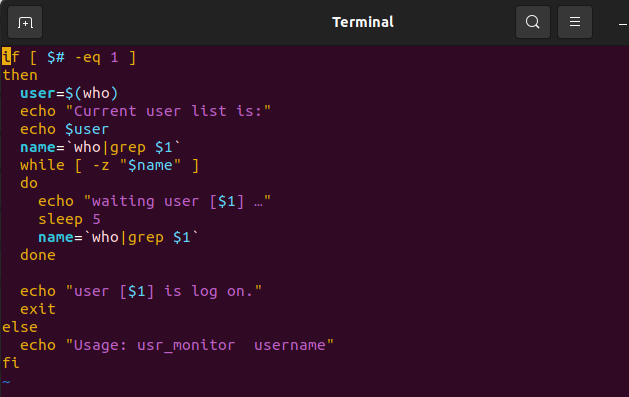
1. 使用while循环，用sleep命令每隔5秒钟检查指定用户是否已登录系统。检测方法与步骤3相同。
2. 如果指定用户未登录，则一直等待下去；如果指定用户已登录，则在省略号后面接着显示：

[username] is log on

退出循环，并结束程序。

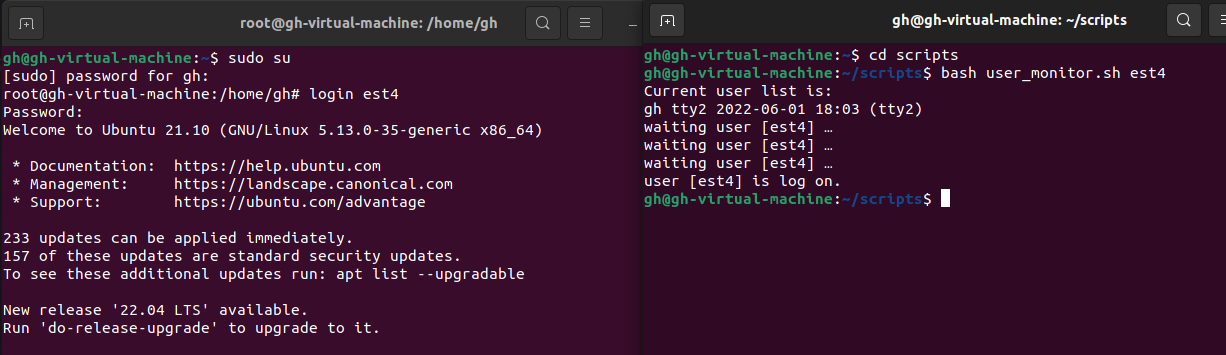
1. 调整程序格式，在程序头加上若干行程序功能注释，在程序的重要功能语句处加上简单注释。

代码如下：

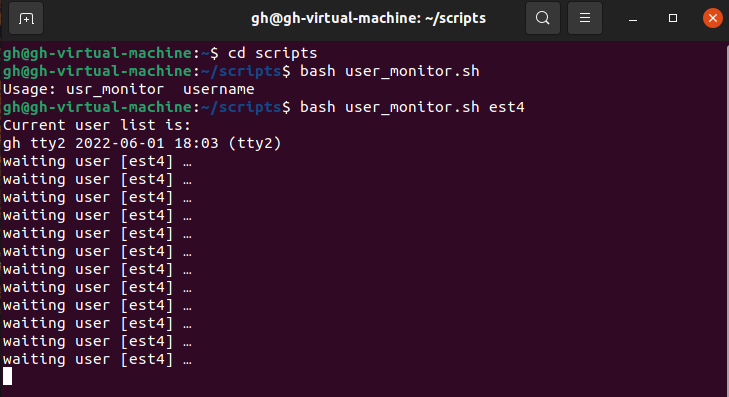


**九、实验数据及结果分析：**

1. usr\_monitor程序运行时，指定est4用户名，程序显示当前已登录系统的用名单，等待指定的est4用户的登录，这个时候会不断地测试est4的登陆情况，当est4登录系统时，程序给出提示信息，并退出。（这个时候需要两个终端，一个终端需要用来进行用户登录的测试，另外一个终端用来进行用户登录）



2. usr\_monitor程序运行时，指定guest用户名，程序显示当前已登录系统的用名单，等待指定的guest用户的登录，若没有登陆，则一直等待。



**十、总结及心得体会：**

在本实验中，需要学会的是如何使用shell脚本程序进行文件状态的测试，同样还熟悉了在Linux的环境下如何进行用户创建，并且进行创建用户的登录，熟悉了Linux的基础命令操作，并且也熟悉了shell脚本程序并且熟悉了文件操作。

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名： 高豪 学 号：2020090915005 指导教师：任立勇**

**实验地点： 信软楼西305 实验时间： 2021.3**

1. **实验室名称：**

信软楼西304

1. **实验项目名称：**

系统命令的实现

1. **实验学时：**

4学时

1. **实验目的：**

掌握Linux目录操作方法，包括打开目录、关闭目录、读取目录文件；掌握Linux文件属性获取方法，包括三个获取Linux文件属性的函数、文件属性解析相关的宏；掌握POSIX与ANSI C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。

1. **实验原理：**

1、遍历目录

目录可以被任何有这个目录读权限的人读。然而只有内核才能写一个目录，来保证文件系统的健全。写权限位和执行权限位决定我们是否可以在该目录创建和删除文件－－他们并没有指定我们是否可以写这个目录本身。

目录的真实格式取决于Linux系统实现和文件系统的设计。许多系统避免程序使用read函数来访问目录的内容，因此更加隔离了程序和目录格式的实现相关的细节。

#include <dirent.h>

DIR \*opendir(const char \*pathname);  
成功返回指针，失败返回null。

struct dirent \*readdir(DIR \*dp);  
成功返回指针，目录尾或失败返回null。

void rewinddir(DIR \*dp);  
int closedir(DIR \*dp);

成功返回0，失败返回-1。

long telldir(DIR \*dp);  
 返回dp相关的目录的当前位置。

void seekdir(DIR \*dp, long loc);

在文件<dirent.h>里定义的dirent结构体依赖于系统实现。实现定义这个结构体至少要包括以下两个成员：  
struct dirent {  
  ino\_t d\_ino;    /\* i-node号 \*/  
  char d\_name[NAME\_MAX + 1]; /\* null结尾的文件名 \*/  
};

DIR结构体是一个内部结构，被那六个函数用来维护关于被读目录的信息。DIR结构体的作用与标准I／O库维护的FILE结构体类似，从opendir返回的DIR结构体的指针被其它5个函数使用。opendir函数初始化所有事情，以便第一个readdir操作读到目录的第一项。在目录里的项的顺序是实现相关的，而且通常不是字母序。

2、Linux文件属性操作

Linux系统的多数文件都是普通文件或者目录，但也有其它的文件类型。文件类型包括：

（1）普通文件。最普遍的文件类型，以某种格式包含数据。对于UNIX内核而言，数据是文本还是二进制没有区别。一个普通文件内容的任何任何解释都留给处理这个文件的程序。一个显著的例外是二进制可执行文件。为了执行一个程序，内核必须了解它的格式。所有的二进制可执行文件都遵守一个允许内核来找到在哪里载入一个程序的代码和数据的格式。

（2）目录文件。一个包含其它文件名和这些文件的信息指针的文件。任何从一个有目录文件读权限的进程都可以读取这个目录的内容，但只有内核才能直接向一个目录文件写入。进程必须使用这章介绍的来对一个目录进行修改。

（3）块特殊文件。一种提供以固定尺寸单位的缓冲I／O方式访问设备（比如硬盘）的文件类型。

（4）字符特殊文件。一种提供以可变尺寸单位的未缓冲I／O方式访问设备的文件类型。在一个系统上的所有设备不是块特殊文件就是字符特殊文件。

（5）FIFO。一种用于进程间通信的文件类型。它有时被称为命令管道。

（6）套接字。一种用于进程间网络通信的文件类型。一个套接字也可以用来在同一主机上的进程间的非网络通信。我们在16章使用套接字来进程进程间的通信。

（7）符号链接。一种指向另一个文件的文件类型。

在Linux中可以通过三个stat函数，来获取文件相关属性信息，包括文件类型、文件权限、文件所属用户、文件最后修改时间等等，如下所示：

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

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

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

三个函数成功都返回0，错误都返回-1。

一个文件的类型编码在stat结构体的st\_mode成员里。我们可以使用下表的宏来决定文件类型。这些宏的参数都是stat结构体的st\_mode成员：

|  |  |
| --- | --- |
| <sys/stat.h>里的文件类型判断宏 | |
| 宏 | 文件类型 |
| S\_ISREG() | 普通文件 |
| S\_ISDIR() | 目录文件 |
| S\_ISCHR() | 字符特殊文件 |
| S\_ISBLK() | 块特殊文件 |
| S\_ISFIFO() | 管道或FIFO |
| S\_ISLNK() | 符号链接 |
| S\_ISSOCK() | 套接字 |

3、Linux文件I/O操作

（1）open函数（open Function）

通过调用open函数可以打开或创建一个文件：

int open(const char \*pathname, int oflga, .../\* mode\_t mode \*/ );

如果成功返回文件描述符，错误返回-1。

pathname是打开或创建的文件的名字。第三个参数为...，是ISO C表示剩余参数的数量和类型可变的方法。对于这个函数来说，第三个参数仅当创建一个文件时使用，我们稍后讨论。我们在函数原型中把这个参数作为一个注释。

这个函数有一个选项群，由参数oflag描述。这个参数由一个或多个以下定义在<fcntl.h>里的常量的通过位或组合而成：

* O\_RDONLY：只读方式打开；
* O\_WRONLY：只写方式打开；
* O\_RDWR：读写方式打开。

为了与早期的程序兼容，多数实现把O\_RDONLY定义为0、O\_WRONLY定义为1、而O\_RDWR定义为2。

这三个常量必须有且只能有一个被指定。以下的常量是可选的：

* O\_APPEND：每次写时添加到文件末尾；
* O\_CREAT：创建这个文件如果它不存在。这个选项需要为open函数提供第三个参数－－打开模式－－来指定这个新建文件的访问权限位
* O\_EXCL：如果O\_CREAT也同时被指定而文件已存在的话会产生一个错误 。这个文件是否存在的测试以及当文件不存在时文件的创建是一个原子操作。
* O\_TRUNC：如果文件存在而且它以只写或读写方式被成功打开的话，把它的长度截为0；
* O\_NOCTTY：如果路径名指向一个终端设备，则不要为这个进程把这个设备作为控制终端分配；
* O\_NONBLOCK：如果文件名指向一个FIFO，一个块特殊文件或一个字符特殊文件，这个选项为打开文件及后续I/O设置非阻塞模式。

（2）creat函数（creat function）

一个新文件也可以通过调用creat函数来创建：

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

成功则返回打开的只写文件描述符，错误返回-1，这个函数等同于：

open (pathname, O\_WRONLY | O\_CREAT | O\_TRUNC, mode);

历史上早期的UNIX版本，open的第二个参数只能是0、1或2。当时不可能用open函数来打开一个不存在的文件。因此，一个独立的系统调用－－creat，被用来创建新的文件。现在O\_CREAT和O\_TRUNC选项可以在open里使用，独立的creat已经不再需要了。

当我们在后面的内容中深入讨论文件访问权限时再来说明如何指定mode。

creat的一个缺点是打开的文件只能用来写。在open的新版本出现之前，如果我们想打开一个临时文件写入然后再读回，我们必须调用creat，close，然后再是open。一个更好的方式是使用open函数：

open (pathname, O\_RDWR | O\_CREAT | O\_TRUNC, mode);

（3）close函数（close function）

一个打开的文件可以通过调用close函数来关闭：

int close(int filedes);

成功返回0，错误返回-1。

关闭一个文件会同时释放进程在这个文件上可能会有的任何记录锁（record locks）。当一个进程终止时，它所有的打开文件都被内核自动关闭。许多程序利用这个便利而不显式地关闭打开的文件。

（4）lseek函数

每个打开的文件都有一个相应的“当前文件偏移量（current file offset）”，一般是一个表示从文件开头开始的字节数量的非负整数。读和写的操作一般从当前文件偏移量开始，而且会根据读出或写入的字节数导致偏移量的增加。当文件打开时，这个偏移量默认初始化为0，除非O\_APPEND选项被指定。

一个打开的文件的偏移量可以通过调用lseek显示地设置：

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

成功返回新的文件偏移，错误返回-1。

offset的解释取决于whence参数：

如果whence为SEEK\_SET，那文件偏移量会设置为从文件开头之后的offset字节数的位置；如果whence为SEEK\_CUR，那文件偏移量会设置为当前的偏移量加上offset的值。offset可以是正也可以是负的；如果whence为SEEK\_END，那文件偏移量为文件的尺寸加上offset的值。offset可以是正也可以是负的。

因为一个成功的lseek调用会返回新的文件偏移量，我们可以通过从当前位置seek 0字节来决定当前的偏移量：

off\_t currpos;

durrpos = lseek(fd, 0, SEEK\_CUR);

这个技术同样也可以用来确定一个文件是否有seek的能力。如果文件描述符指向一个管道、FIFO或套接字，lseek把errno设置为ESPIPE并返回1。

通常情况下，一个文件的当前偏移量必须是非负整数。尽管如此，有些设备可能允许负的偏移量。但对于普通文件，偏移量必须是非负的。因为负数偏移量是可能的，我们应该小心比较lseek的返回值，应测试它是否等于-1，而不能测试它是否小于0。

（5）read函数（read Function）

可以使用read函数从打开的文件里读取数据：

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

返回读取到的字节数，文件结尾则返回0，错误返回-1。以下几种情况会造成实际读取的字节数比请求的数量少：

在读一个普通文件时，如果在读入请求的字节数之前碰到了文件尾。比如，在请求读取100字节时，只剩30字节就到文件末尾了，read返回30。下次我们调用read时，它会返回0（文件末尾。）

当从一个终端设备读数据时。通常，一次最多只能读一行。（我们将在18章展示如何改变它。）

当从网络上读数据时。网络的缓冲区可能会导致读入的比请求的少。

当从一个管道或FIFO中读数据时。如果管道包含比请求量少的字节，read会只返回可用的字节数。

当从一个面向记录（record-oriented）设备读入数据时。一些面向记录的设备，比如磁带（magnetic type），一次可以最多返回一个记录。

当被一个信号中断，而只读了部分数据时

read操作从文件当前偏移量开始。在成功返回前，偏移量会增加实际读取的字节数。

（6）write函数（write Function）

可能用write函数向一个打开的文件里写数据：

#include <unistd.h>

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

成功返回写入的字节数，错误返回-1。返回值一般与nbytes参数相等，不然就说明出错了。write错误的一个常见的原因可能是磁盘写满或超过了一个进程的文件大小限制。

对于一个普通文件，写操作从文件当前偏移量开始。如果当文件被打开时O\_APPEND选项被指定，每次写操作前，文件的当前偏移量会设在在文件末尾。在成功写入后，文件偏移量会增加真正写入的字节数。

1. **实验内容：**

利用POSIX API（文件操作也可以使用ANSI C标准I/O库）编程实现cp –r命令，支持将源路径（目录）中的所有文件和子目录，以及子目录中的所有内容，全部拷贝到目标路径（目录）中。

1. **实验器材（设备、元器件）：**

PC机， Ubuntu系统

1. **实验步骤：**

1. 总体程序流程：

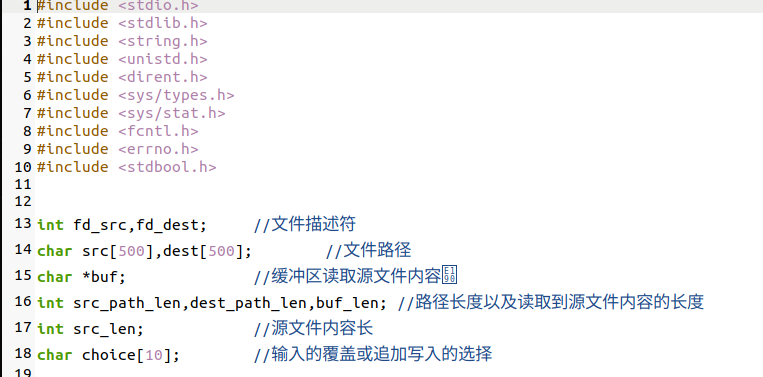
图示

描述已自动生成

2. 文件到文件拷贝



1. 代码展示如下：



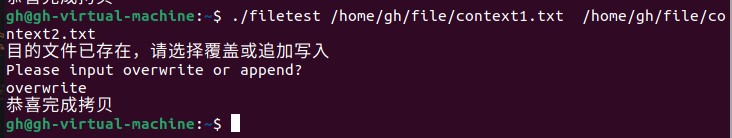






1. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**

1. 文件到文件



源文件内容：

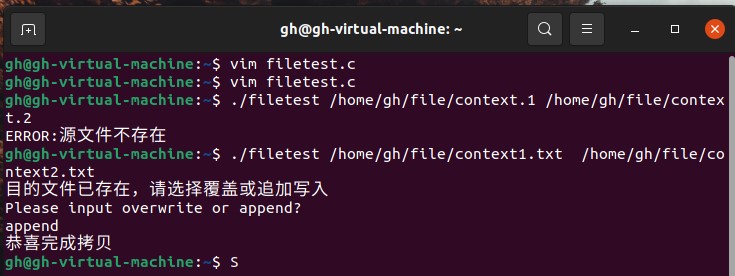




拷贝文件内容为：



文件追加：

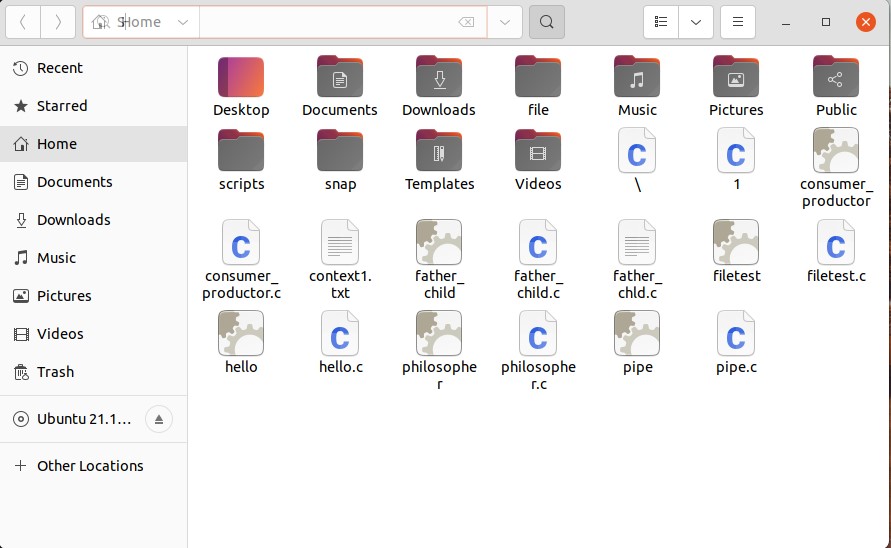




2. 文件到目录：

文件到目录拷贝

复制后目录内容：



1. **总结及心得体会：**

在本次实验中，学会了如何使用Linux命令进行文件的复制、拷贝操作，同时明白了在C语言中如何进行相应的文件操作，熟悉了Linux和C两种命令的操作，让自己对整个操作更加熟悉流畅

**报告评分：**

**指导教师签字：**