苏州大学实验报告

院、系	计算机学院	年级专业	20 计科	姓名	柯骅	学号	2027405033
课程名称	操作系统课程实践					成绩	
指导教师	李培峰	同组实验者	无		实验日期	2023.03.23	

实 验 名 称 实验 4 简单文件系统设计

一、 实验目的

1. 熟悉 Linux 文件系统的文件和目录结构。
2. 掌握文件系统的基本特征。
3. 掌握常用的文件操作函数。
4. 理解文件存储空间的管理、文件的物理结构和目录结构以及文件操作的实现。
5. 加深对文件系统内部功能和实现过程的理解。

二、 实验内容

1. （文件备份实验）

编写 C 程序，模拟实现 Linux 文件系统的简单 I/O 流操作：

备份文件，将源文件 source.dat 备份为 target.dat 文件。要求：

(1) 使用 C 库函数实现文件备份

(2) 使用系统调用函数实现文件备份

2. （简单文件系统的模拟）

模拟实现一个简单的二级文件管理系统，要求做到以下几点：

(1) 可以实现常用文件目录和文件操作，如：

`login` 用户登录　`dir` 列文件目录　`create` 创建文件　`delete` 删除文件

`open` 打开文件　`close` 关闭文件　`read` 读文件　`write` 写文件

(2) 列目录时要列出文件名、物理地址、保护码和文件长度

(3) 源文件可以进行读写保护

三、 实验步骤

1. （文件备份实验）

(1) 使用 C 库函数实现文件备份

C 语言库函数：

fopen()函数

```
1. FILE *fopen(const char *filename, const char *mode)
```

作用：使用给定的模式 mode 打开 filename 所指向的文件。

参数：filename -- 字符串，表示要打开的文件名称。

mode ------ 字符串，表示文件的访问模式，可以是以下表格中的值：

模式	描述
"r"	打开一个用于读取的文件。该文件必须存在。
"w"	创建一个用于写入的空文件。如果文件名称与已存在的文件相同，则会删除已有文件的内容，文件被视为一个新的空文件。

教务处制

"a"	追加到一个文件。写操作向文件末尾追加数据。如果文件不存在，则创建文件。
"r+"	打开一个用于更新的文件，可读取也可写入。该文件必须存在。
"w+"	创建一个用于读写的空文件。
"a+"	打开一个用于读取和追加的文件。

fclose()函数

```
1. int fclose(FILE *stream)
```

作用 ：关闭流 stream。刷新所有的缓冲区。

参数 ：stream -- 这是指向 FILE 对象的指针，该 FILE 对象指定了要被关闭的流。

返回值：如果流成功关闭，则该方法返回零。如果失败，则返回 EOF。

fread()函数

```
1. size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
```

作用：从给定流 stream 读取数据到 ptr 所指向的数组中。

参数：

ptr ------ 指向带有最小尺寸 size*nmemb 字节的内存块的指针。

size ---- 要读取的每个元素的大小，以字节为单位。

nmemb --- 元素的个数，每个元素的大小为 size 字节。

stream -- 指向 FILE 对象的指针，该 FILE 对象指定了一个输入流。

返回值：

成功读取的元素总数会以 size_t 对象返回，size_t 对象是一个整型数据类型。如果总数与 nmemb 参数不同，则可能发生了一个错误或者到达了文件末尾。

fwrite()函数

```
1. size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)
```

作用：把 ptr 所指向的数组中的数据写入到给定流 stream 中。

参数：

ptr ----- 这是指向要被写入的元素数组的指针。

size ---- 这是要被写入的每个元素的大小，以字节为单位。

nmemb --- 这是元素的个数，每个元素的大小为 size 字节。

stream -- 这是指向 FILE 对象的指针，该 FILE 对象指定了一个输出流。

返回值：

如果成功，该函数返回一个 size_t 对象，表示元素的总数，该对象是一个整型数据类型。如果该数字与 nmemb 参数不同，则会显示一个错误。

具体步骤：

a) 使用 fopen()函数以只读方式打开想要备份的源文件 source,并以只写方式打开想要写入内容的目标文件 target,代码如下,这里的 r 和 w 分别表示只读和只写模式。在创建完 source 和 backup 变量后，还需要判断他们是否为 NULL，来验证创建是否成功。

```
1. source=fopen("./source.dat","r");
2.  backup=fopen("./target.dat","w");
```

b) 使用 fread()函数循环读取源文件中一个缓冲区大小的内容,然后使用 fwrite()函数将读取的内容写入目标文件。这里 fread()函数表示从给定流 source 中读取一个大小为 sizeof(buf)的元

素（即一个字符），并将其赋值给 buf 变量。fwrite()表示将一个大小为 sizeof(buf)的元素（即一个字符）写入 backup 流中进行存储。

```
1. while(fread(&buf,sizeof(buf),1,source)==1){
2.     if(!fwrite(&buf,sizeof(buf),1,backup)){    ...    }    }
```

c）读取与写入完毕后，使用 fclose()函数关闭读写文件流。同样检查文件流是否正常关闭。

```
1. if(fclose(source))
2. if(fclose(backup))
```

(2) 使用系统调用函数实现文件备份

Linux 系统调用函数：

open()函数：

作用：打开或创建一个文件，在打开或创建文件时可指定文件的属性及用户权限等。

头文件及原型：

```
1. #include <sys/types.h>
2. #include <sys/stat.h>
3. #include <fcntl.h>
4. int open(const char *pathname, int flags, mode_t mode);
```

参数：

序号	参数	描述
1	pathname	要打开或创建的文件名，和 fopen 一样，pathname 既可以是 相对路径 也可以是 绝对路径
2	flags	有一系列常数值可供选择，可以同时选择多个常数用 按位或运算符 | 连接起来，所以这些常数的宏定义都以 O_ 开头，表示 or

序号	常数	描述
1	O_RDONLY	只读打开
2	O_WRONLY	只写打开
3	O_RDWR	可读可写打开

返回值：成功返回新分配的文件描述符，出错返回-1

close()函数：

作用：关闭一个已打开的文件

头文件及原型：

```
1. #include <unistd.h>
2. int close(int fd);
```

参数 fd 是要关闭的文件描述符。需要说明的是，当一个进程终止时，内核对该进程所有尚未关闭的文件描述符调用 close()关闭，所以即使用户程序不调用 close()，在终止时，内核也会自动关闭它打开的所有文件。但是对于一个长年累月运行的程序（比如 网络服务器），打开的文件描述符一定要记得关闭，否则随着打开的文件越来越多，会占用大量文件描述符和系统资源。

返回值：成功返回 0，出错返回 -1

read()函数：

作用：从打开的设备或文件中读取数据

头文件及原型：

```
1. #include <unistd.h>
2. ssize_t read(int fd, void *buf, size_t count);
```

参数 count 是请求读取的字节数，读上来的数据保存在缓冲区 buf 中，同时文件的当前读写位置向后移。注意这个读写位置和使用 C 标准 I/O 库时的读写位置有可能不同，这个读写位置是记在内核中的，而使用 C 标准 I/O 库时的读写位置是用户空间 I/O 缓冲区中的位置。比如用 fgetc 读一个字节，fgetc 有可能从内核中预读 1024 个字节到 I/O 缓冲区中，再返回第一个字节，这时该文件在内核中记录的读写位置是 1024，而在 FILE 结构体中记录的读写位置是 1。

返回值：成功返回读取的字节数，出错返回-1，如果在调 read()之前已到达文件末尾，则这次 read()返回 0。

write()函数：

作用：向打开的设备或文件中写数据

头文件及原型：

```
1. #include <unistd.h>
2. ssize_t write(int fd, const void *buf, size_t count);
```

返回值：成功返回写入的字节数，出错返回-1 并设置 errno。

具体步骤：

(1) 使用 open()系统调用函数以只读方式打开想要备份的源文件 source,并以只写方式打开想要写入内容的目标文件 target。

```
1. source=open("./source.dat",O_RDONLY);
2. backup=open("./target.dat",O_WRONLY|O_CREAT,0644);
```

这里的 O_RDONLY 表示只读，O_WRONLY|O_CREAT 表示如果没有该文件则创建并且只写。

(2)使用 read()系统调用函数循环读取源文件中一个缓冲区大小的内容,然后使用 write()系统调用函数将读取的内容写入目标文件。

```
1. while((size=read(source,buf,MAXSIZE))>0){
2.     if(write(backup,buf,size)!=size){    ...    } }
```

(3)读取与写入完毕后，使用 close()系统调用函数关闭读写文件流。

2. （简单文件系统的模拟）

设计思路

本文件系统采用两级目录，其中第一-级对应用户账号，第二级对应用户账号下的文件。由于只进行简单的模拟实现，因此不考虑文件共享、文件系统安全及特殊文件等内容。设计时，首先应确定文件系统的数据结构，即主目录、子目录和活动文件等。主目录和子目录都以链表的形式存放，用户创建的文件以编号形式存储在磁盘上，并且需要在目录中进行登记。

设计数据结构

在实现这个文件系统时需要设计如下数据结构：

（1）磁盘块结构体（disTable）

对于磁盘块结构体，其中需包含：磁盘块容量(maxlength)，起始地址(start)，是否被使用(useflag)，指向下一个磁盘块的指针(next)这几个参数。

（2）文件块结构体（FCB，命名为 fileTable）

对于文件块结构体，其中需包含：文件名数组(fileName[10])，文件在磁盘中的起始地址(start)，文件内容长度(length)，文件的最大长度(maxlength)，文件的属性(fileKind[3])，文件的相关时间信息(timeinfo)，是否已被打开标志(openFlag)。

（3）用户文件目录（User file directory，UFD）

对于用户文件目录，其中需包含：指向文件块结构体的指针(file)，下一个用户文件目录指针(next)。

（4）主文件目录（master file directory，MFD）

对于主文件目录，其中需包含：用户账号数组(userName[10]),用户密码(password[10])，指向用户文件目录的指针(user)。

具体功能

login:

该指令用于进入已经创建的用户空间，格式：login [user]

具体来说，首先读取用户希望登录的用户名，把它存在字符数组 name 中，然后在已经注册好的用户表 userTable 里寻找用户是否存在，若不存在则给出报错，若存在则给 3 次输入密码的机会，同样读入密码后比对，若三次密码皆为错误密码，则退出此模拟程序，表示系统锁定。

dir:

该指令用于显示某一用户的所有文件，格式：dir [user]

具体来说，首先找出某一用户在用户表 userTable 中的下标，如果不在用户表中，则表示用户不存在并输出，如果存在，那么取出用户文件链表的首位 userTable[k].user->next，依次按照链表顺序查找并按照要求格式输出。

create：

该指令用于为某一用户创建文件，格式：create [fileName] [fileLength] [authority]，三个参数分别表示文件名，文件长度和文件权限。

具体来说，接收到指令后，先判断文件是否重名，如果不重名，则使用 malloc 向内存申请相应空间，这里要注意，fileNode 里面的指针也需要申请地址，同时存储文件相应参数，将创建好的文件空间插入到用户空间链表的末尾。

delete：

该功能主要由 rm 指令完成，用于删除指定文件，格式：rm [fileName]

具体来说，在接收到指令后，首先在用户空间链表中寻找是否存在相应文件名的文件，如果不存在，那么输出错误信息；如果存在，那么判断该文件是否已经被打开。如果已经被打开，说明有进程正在占用该文件，我们需要等待进程取消占用才可以对它进行删除操作；如果没有被打开，那么可以顺利删除，将链表中该文件前后两端相连，并使用 free 函数释放相应空间即可。

open：

该功能主要用于实现进程互斥，在本系统中任何文件只能同时被一个进程占用。

具体来说，我们为每一个文件设置一个 openFlag，表示该进程是否被占用，在创建文件时，openFlag 初始化为 false，表示没有进程占用；当使用 cat 或 write 指令修改文件时，则需要指定文件的 openFlag 为 false 状态才可以执行，同时执行过后需要将 openFlag 标记为 true，表示当前文件正在被占用；当使用 rm 指令时，则需要文件的 openFlag 为 false，表示当前文件没有被占用，可以正常删除。

close：

该功能与 open 对应，用于将 openFlag 置为 false，格式：close [fileName]

具体来说，当我们使用 cat 或 write 指令后，文件的 openFlag 被置为 true，我们需要先使用 close 将文件关闭才可以对文件进行删除等操作。

read：

该功能主要由 cat 指令完成，用于查看文件，格式：cat [fileName]

具体来说，首先在用户的文件空间中寻找指定文件，如果不存在，则输出错误；如果存在，那么找到文件在内存中的位置，按照字符一个个输出，每 50 个字符换一次行，同时将 openFlag 置为 true，表示当前文件已经被 cat 指令打开占用。

write：

该指令主要用于写入文件，格式：write [fileName]

具体来说，首先需要判断该文件是否存在并且是否有写入权限，如果不存在或没有写入权限，那么输出错误信息退出；如果存在并且有写入权限，那么读取文本需要写入的内容，如果内容超过了该文件的最大容量，那么输出错误信息退出；如果内容没有超过最大容量，那么写入成功，将 openFlag 置为 true，表示该文件已经被打开占用。

四、 实验结果

1. （文件备份实验）

(1) 使用 C 库函数实现文件备份

执行后，发现两个文件大小相等，备份成功

(2) 使用系统调用函数实现文件备份

执行后，发现两个文件大小相等，备份成功

2. （简单文件系统的模拟）

创建一个用户名为“user1”密码为“123456”的用户并登录结果如下：

创建最大容量为 1000，权限为 rw 的文件 a 和最大容量为 1000，权限为 r 的文件 b 结果如下：

```
create-创建 格式: create a1 1000 rw,将创建名为a1,长度为1000字节可读可写的文件
rm-删除 格式: rm a1,将删除名为a1的文件
cat-查看文件内容 格式: cat a1,显示a1的内容
write-写入  格式: write a1
fine-查询 格式: fine a1 ,将显示文件 a1的属性
chmod-修改 格式: chmod a1 r,将文件a1的权限改为只读方式
ren-重命名 格式: ren a1 b1 ,将a1改名为b1
dir-显示文件 格式: dir aaa,将显示aaa用户的所有文件
df-显示磁盘空间使用情况 格式: df
close-关闭文件 格式: close a1,将关闭文件a1
return-退出用户，返回登录界面
exit-退出程序
please imput your command:>create a 1000 rw
创建文件成功
```

```
please imput your command:>create b 1000 r
创建文件成功
```

在 a 文件中写入内容“666”，并关闭 a 文件结果如下：

```
please imput your command:>write a
please input content:
666
文件写入成功,请用close命令将该文件关闭
```

```
please imput your command:>close a
a文件已关闭
```

查看 a 文件并关闭，结果如下：

```
please imput your command:>cat a
**************************************************

666

**************************************************
a已被read进程打开,请用close命令将其关闭
```

```
please imput your command:>close a
a文件已关闭
```

将 a 文件权限修改为 r，并查询，结果如下：

```
please imput your command:>chmod a r
修改文件类型成功
```

```
please imput your command:>fine a
******************************************
文件名: a
文件长度: 1000
文件在存储空间的起始地址: 0
文件类型: r
创建时间: Wed Mar 29 06:47:15 2023

******************************************
```

将文件 a 重命名为 c，并查看 user1 的所有文件，结果如下：

```
please imput your command:>ren a c
重命名成功
```

```
please imput your command:>dir user1
**************************************************
文件名   文件长度   文件在磁盘的起始地址   文件类型   创建时间
c        1000            0            r    Wed Mar 29 06:47:15 2023
b        1000            0            r    Wed Mar 29 06:47:15 2023
**************************************************
```

删除文件 c（原文件 a）后，查询磁盘空间使用情况，结果如下：

```
please imput your command:>rm c
文件删除成功
```

```
please imput your command:>dir user1
**************************************************
文件名   文件长度   文件在磁盘的起始地址   文件类型   创建时间
b        1000            0            r    Wed Mar 29 06:47:15 2023
**************************************************
```

```
please imput your command:>df
************************************************************************
  盘块号     起始地址       容量(byte)   是否已被使用
   0          2000            522288          0
   1          0              1000          0
   2          1000            1000          1
************************************************************************
磁盘空间总容量: 512*1024Bytes  已使用: 1000Bytes   未使用: 523288Bytes
```

五、 实验思考与总结

1. （文件备份实验）

（1）使用系统调用函数open()、read()、write()和close()实现简单文件备份的原理是什么？

首先分别打开源文件（只读方式）和目标文件（只写方式）。对于打开的源文件，每次读1个字节，然后写入目标文件，如此循环，直到文件结束。最后关闭两个文件。

（2）使用C语言库函数fopen()、fread()、fwrite()和fclose()实现简单文件备份的原理是什么？

首先分别打开源文件（只读方式）和目标文件（只写方式）。对于打开的源文件，每次读1024个字节（示例代码中buf的长度），然后写入目标文件，如此循环，直到文件结束。最后关闭两个文件。

（3）上述两种方式的区别是什么？

这两种方法基本是类似的，主要区别是一个是Linux操作系统提供的系统调用，一个C语言提供的库函数；另外，从示例代码看使用C语言库函数进行文件备份的速度较快。

2. （简单文件系统的模拟）

（1）说明在文件系统提供与不提供打开操作的情况下，读写文件时的不同。

当系统没有显式提供打开文件的操作时，需要每次读文件或写文件时，都需要判断该文件是否已经打开，如果未打开，则打开该文件，否则就直接执行读写操作。

3. 实验总结

本次是简单文件系统设计的实验。在文件备份实验中，使用了 Linux 操作系统提供的系统调用和 C 语言提供的库函数，两种方法实现了对于 source.bat 文件的备份，学习并使用了 fopen()open()等函数，能够对文件进行基础操作。

在简单系统的模拟的实验中，使用 C 语言设计了一系列结构体用于实现简单系统，例如磁盘块结构体，文件块结构体以及用户文件目录等，实现了 login，dir，create，rm 等指令的模拟，对于文件管理系统有了进一步的了解，加深了对文件系统内部功能和实现过程的理解。