**《操作系统课程设计》**

**实验报告**

徽标, 公司名称

描述已自动生成

**题目: 文件访问**

**学 院 计算机学院**

**班 级 2021211304**

**学 号 2021212171**

**姓 名 杨晨**

**2024年 4月**

## 目 录

第一章 实验概述 1

1.1 实验目的 1

1.2 实验内容 1

1.2.1 XXX 1

1.2.2 XXXX 1

1.3 XXXX 1

第二章 实验步骤 2

2.1 XXXX 2

2.1.1 XXXX 2

2.1.2 XXX 2

2.2 XXXX 2

2.2.1 XXX 2

2.2.2 XXX 2

2.3 XXXXX 10

2.3.1 XXXX 10

2.4 XXXXX 12

2.4.1 XXXX 12

2.4.2 XXXX 16

第三章 实验结果分析 22

3.1 XXX 22

3.1.1 XXXX 22

3.1.2 XXXX 22

3.2 XXXXX 22

3.2.1 XXX 22

3.2.2 XXX 24

3.3 XXXX 24

3.3.1 XXXX 24

3.3.2 XXX 26

3.4 XXX 26

3.4.1 XXXXX 26

3.4.2 XXXX 29

3.4.3 XXXX 29

第四章 总结 36

4.1XXXXX 36

# **第一章 实验概述**

##### 1.1 实验目的

利用create、open、close、read、write等文件访问系统调用，将创建文件，并源文件中内容拷贝至目标文件

##### 1.2 实验内容

# **1.2.1 open()函数**

**功能描述**：用于打开或创建文件，在打开或创建文件时可以指定文件的属性及用户的权限等各种参数

**所需头文件**：#include <sys/types.h>, #include <sys/stat.h>,#include <fcntl.h>

**函数原型**：int open(const char \*pathname, int flags, int perms)

**参数**：

(1) pathname: 被打开的文件名（可包括路径名，如"dev/ttyS0"）

(2) flags：文件打开方式

O\_RDONLY：以只读方式打开文件；

O\_WRONLY：以只写方式打开文件；

O\_RDWR：以读写方式打开文件；

O\_CREAT：如果打开的文件不存在，创建一个新的文件，并用第三个参数为其设置权限；

O\_EXCL：如果使用O\_CREAT 时文件存在，则返回错误消息。这一参数可测试文件是否存在。此时 open 是原子操作，防止多个进程同时创建同一个文件；

O\_NOCTTY：使用本参数时，若文件为终端，那么该终端不会成为调用 open()的那个进程的控制终端；

O\_TRUNC：若文件已经存在，那么会删除文件中的全部原有数据，并且设置文件大小为 0；

O\_APPEND：以添加方式打开文件，在打开文件的同时，文件指针指向文件的末尾，即将写入的数据添加到文件的末尾；

O\_NONBLOCK：如果 pathname 指的是一个FIFO、一个块特殊文件或一个字符特殊文件，则此选择项为此文件的本次打开操作和后续的 I/O 操作设置非阻塞方式；

O\_SYNC：使每次write 都等到物理I/O 操作完成；

O\_RSYNC：read 等待所有写入同一区域的写操作完成后再进行；在open()函数中，falgs 参数可以通过“|”组合构成，但前 3 个标准常量（O\_RDONLY，

O\_WRONLY，和O\_RDWR）不能互相组合。

(3) perms：被打开文件的存取权限，可以用两种方法表示，可以用一组宏定义：

S\_I(R/W/X)(USR/GRP/OTH)，其中R/W/X 表示读写执行权限，

USR/GRP/OTH 分别表示文件的所有者/文件所属组/其他用户，如S\_IRUUR|S\_IWUUR|S\_IXUUR,（-rex------）；

也可用八进制800 表示同样的权限

**返回值**：

成功：文件描述符 fd

失败：-1

当前文件偏移量cfo(current file offset) in Linux：

cfo 表示文件开始处到文件当前位置的字节数，一般是一个非负整数。对文件的 read()、write()通常始于cfo，即根据 cfo 指向的文件位置，进行读写操作，并使cfo 值增大，cfo 的增量为读写的字节数。

使用 open()操作打开文件时，文件的 cfo 初始化为 0，指向文件开始位置，除非参数flags=O\_APPEND；使用系统调用 lseek()可以改变文件的 cfo，即改变文件读写指针位置。

# **1.2.2 close()函数**

**功能描述**：用于关闭一个被打开的的文件

**所需头文件**： #include <unistd.h>

**函数原型**：int close(int fd)

**参数**：fd 文件描述符

**函数返回值**：0，成功，-1，出错

# **1.2.3 read()函数**

**功能描述**：从文件读取数据

**所需头文件**：#include <unistd.h>

**函数原型**：ssize\_t read(int fd, void \*buf, size\_t count)

**参数**：

(1) fd：读取数据的文件的文件描述符；

(2) buf：内存缓冲区，用于存放从文件读取的数据；

(3) count：执行一次 read 操作，应该读取的字节数；

**返回值**：

返回所读取的字节数；

0，读到文件尾部 EOF；-1，出错。

读操作从指针cfo 处开始，在成功返回之前，cfo 增加，增量为实际读取到的字节数。

以下下述情况将导致读取到的字节数小于 count：

A. 读取普通文件时，读到文件末尾时已读取的字节数还不够 count。例如，文件只有 30bytes，而count=100 bytes，那么实际读到的只有 30 bytes，read()返回 30;

B. 从终端设备(terminal device)读取时，一般情况下每次只能读取一行；

C. 从网络读取时，网络缓存可能导致读取的字节数小于 count 字节；

D. 读取pipe 或者FIFO 时，pipe 或FIFO 中的字节数可能小于 count；

E. 从面向记录(record-oriented)的设备读取时，某些面向记录的设备（如磁带）每次最多只能返回一个记录。

F. 在读取了部分数据时被信号 signal 中断。

# **1.2.4 write()函数**

**功能描述**：向文件写入数据

**所需头文件**：#include <unistd.h>

**函数原型**：ssize\_t write(int fd, void \*buf, size\_t count)

**参数**：

(1) fd：写入数据的文件的文件描述符；

(2) buf：内存缓冲区，用于存放写入文件的数据；

(3) count：调用write 操作，应该写入文件的字节数。

**返回值**：

成功：写入文件的字节数；

失败：-1，当磁盘空间满，或者写入数据后超过文件大小限制。

对于普通文件，写操作始于文件读写指针cfo 。如果打开文件时使用了O\_APPEND（从尾部追加写入），则每次写操作都将数据写入文件末尾。成功写入后，cfo 增加，增量为实际写入的字节数。

# **1.2.5 lseek()函数**

**功能描述**： 在文件描述符 fd 指定的文件中，将文件读写指针cfo 定位到相应位置

**所需头文件**： #include <unistd.h>，#include <sys/types.h>

**函数原型**：off\_t lseek(int fd, off\_t offset, int whence)

**参数**：

(1) fd：文件描述符；

(2) offset：偏移量，读写操作所需要移动的距离，单位是字节byte。值为正，指针移向文件尾部；值为正，指针移向文件头部；

(3) whence：定位参照点

SEEK\_SET：参照点为文件开始处0，定位后 cfo 新位置为偏移量offset，即 cfo=offset；SEEK\_CUR：参照点为读写指针cfo 的当前位置，定位后cfo 新位置为其当前位置加上偏移量offset，即cfo=cfo+offset；

SEEK\_END：参照点为文件结尾，新位置为文件大小加上偏移量offset，即cfo=size-of(fd)+offset。

**返回值**：

成功：cfo 的新位置，即返回当前读取位置在文件中的绝对位置

失败：-1

# **第二章 示例代码**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <errno.h>

#define BUFFER\_SIZE 128 // 每次读写缓存大小，影响运行效率

#define SRC\_FILE\_NAME "src\_file.txt" // 源文件名

#define DEST\_FILE\_NAME "dest\_file.txt" // 目标文件名

#define OFFSET 0 // 文件指针偏移量

int main()

{

int src\_file, dest\_file; // 源文件和目标文件描述符

unsigned char src\_buff[BUFFER\_SIZE]; // 读缓存

unsigned char dest\_buff[BUFFER\_SIZE]; // 写缓存

int real\_read\_len = 0; // 实际读取长度

char str[BUFFER\_SIZE] = "this is a test about\nclose()\nwrite()\nread()\nsleek()\nend of the file\n";

// 创建源文件

src\_file = open(SRC\_FILE\_NAME, O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);

if (src\_file < 0)

{

printf("open file error!!!\n");

exit(1);

}

// 向源文件写入数据

write(src\_file, str, strlen(str));

// 创建目标文件

dest\_file = open(DEST\_FILE\_NAME, O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);

if (dest\_file < 0)

{

printf("open file error!!!\n");

exit(1);

}

lseek(src\_file, OFFSET, SEEK\_SET); // 将源文件的读写指针移动到起始位置

while ((real\_read\_len = read(src\_file, src\_buff, sizeof(src\_buff))) > 0)

{

printf("src\_buff: %s", src\_buff);

write(dest\_file, src\_buff, real\_read\_len);

}

lseek(dest\_file, OFFSET, SEEK\_SET); // 将目标文件的读写指针移动到起始位置

while ((real\_read\_len = read(dest\_file, dest\_buff, sizeof(dest\_buff))) > 0)

; // 读取目标文件内容

printf("dest\_buf: %s", dest\_buff);

close(src\_file);

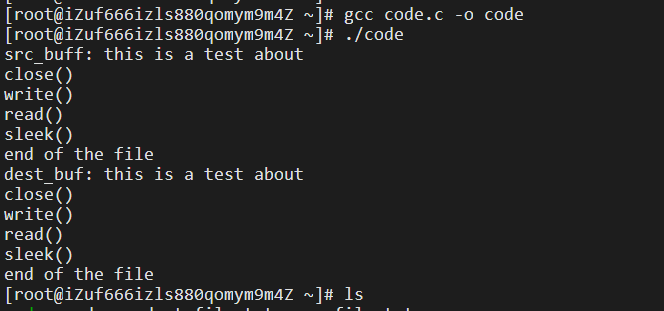
close(dest\_file);

return 0;

}

# **第三章 实验结果分析**

##### 3.1 实验结果



##### 3.2 结果分析

代码的主要目标是从一个源文件中读取数据，并将这些数据写入到一个目标文件中。然后，它再次从目标文件中读取数据，并将这些数据打印到控制台。

以下是详细的运行结果分析：

1. 首先，程序创建了一个名为 "src\_file.txt" 的源文件，并向其中写入了一段字符串。这段字符串包含了几行文本，每行文本之间用换行符 (`\n`) 分隔。

2. 然后，程序创建了一个名为 "dest\_file.txt" 的目标文件。

3. 接下来，程序将源文件的读写指针移动到了文件的起始位置，然后开始读取源文件的内容。每次读取的数据量为 `BUFFER\_SIZE`（在这个例子中，`BUFFER\_SIZE` 的值为 128）。读取的数据被存储在 `src\_buff` 中，并被打印到控制台。

4. 程序将读取到的数据写入到目标文件中。这个过程会一直持续，直到源文件的所有数据都被读取并写入到目标文件中。

5. 然后，程序将目标文件的读写指针移动到了文件的起始位置，然后开始读取目标文件的内容。每次读取的数据量同样为 `BUFFER\_SIZE`。读取的数据被存储在 `dest\_buff` 中，但在这个过程中，程序并没有将读取到的数据打印到控制台。

6. 最后，程序打印了 `dest\_buff` 中的内容。由于 `dest\_buff` 中的内容是从目标文件中读取的，所以这些内容应该与源文件中的内容相同。

7. 程序关闭了源文件和目标文件，并结束了运行。

从运行结果来看，程序的运行结果与预期的行为是一致的。源文件和目标文件中的内容都被正确地打印到了控制台。

##### 参考文献

[1] 附件1-7-1 linux文件访问系统调用（open,close,read,write,lseek)