CS302
Operating System
Lab 10

File System

May 23th , 2018

Xiang Long

File Operations

- A file is an **abstract data type**. To define a file properly, we need to consider the operations that can be performed on files.
  - Create
  - Write - at **write pointer** location
  - Read - at **read pointer** location
  - Repositioning within a file **-seek**
  - Delete
  - Truncate
- These six basic operations comprise the minimal set of required file operations.

open

- 进程访问文件数据前必须先“打开”文件

```
f = open(name, flag);
...
read(f, ...);
...
close(f);
```

- 内核跟踪进程打开的所有文件
  - 操作系统为每个进程维护一个打开文件表(open-file table)
  - 打开文件表通过一个非负整数索引，此非负整数称为文件描述符(file descriptor)
  - 打开一个文件会返回一个“文件描述符”，而随后的操作（读取，写入等）则会以“文件描述符”作为主要参数。

open()系统调用

- 使用open()系统调用打开文件后，会得到一个文件描述符

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open (const char *name,  int flags);
int open (const char *name,  int flags,  mode_t mode);
```

- open()系统调用将路径名称name所指定的文件映射至一个文件描述符，并与映射成功后返回该文件描述符。

flags参数

- flags的参数一个标志位或多个标志位组合而成。它支持三种访问模式(flags参数值中必须包含如下三个标志之一)：O_RDONLY、O_WRONLY或O_RDWR，这三种模式分别表示以只读、只写或读写模式打开文件

```
int fd;

fd  =  open (“/home/cs302/test.txt”,  O_RDONLY);
if (fd != -1){
        printf(“fd is %d\n”, fd);
}
else {
        printf(“open fail\n”);
}
```

flags参数组合

- 在设定flags参数时，可以与下列标志以按位或的方式组合在一起
- O_APPEND
  - 文件将以追加模式打开。也就是说，在每次写操作之前，将会更新文件位置指针，指向文件末尾
- O_CREAT
  - 当参数name指定的文件不存在时，内核自动创建。
- O_EXCL
  - 当和标志位O_CREAT一起使用时，如果参数name指定的文件已经存在，会导致open()调用失败。
- O_DIRECT
  - 打开文件用于直接I/O。
- O_TRUNC
  - 如果文件存在，且是普通文件，并且有写权限，该标志位会把文件长度截断为0。

flags参数组合

```
int fd;

fd  =  open (“/home/cs302/test1.txt”,  O_WRONLY |
          O_TRUNC);
if (fd != -1){
          printf(“fd is %d\n”, fd);
}
else {
          printf(“open fail\n”);
}
```

mode_t 参数

- 除非是在创建文件，否则mode参数会被忽略。如果制定了O_CREAT 标志，则需使用mode参数，如果忘了mode参数，其结果未定义，通常会带来麻烦。

- S_IRUSR -文件所有者有读权限。
- S_IWUSR -文件所有者有写权限。
- S_IRGRP -组用户有读权限。
- S_IWGRP -组用户有写权限。
- S_IROTH -任何人都有读权限。
- S_IWOTH -任何人都有写权限。

使用数字mode参数

- read permission: 4
- write permission: 2

```
fd = open (file, O_WRONLY | O_CREAT | O_TRUNC,
           0664);
```

- 与下面代码等价

```
fd = open (file, O_WRONLY | O_CREAT | O_TRUNC,
           S_IWUSR | S_IRUSR | S_IWGRP | S_IRGRP |
           S_IROTH);
```

creat()系统调用

- 由于O_WRONLY | O_CREAT | O_TRUNC 的组合太常见，所以有一个系统调用专门提供此行为。

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int create (const char  *name,  mode_t  mode);
```

creat()系统调用

- 下面是典型的creat()调用

```
int fd;

fd  =  creat (file,  0644);
if (fd != -1){
        printf("open fail\n");
}
```

- 与下面程序代码功能相同

```
int fd;

fd  =  open (file,   O_WRONLY | O_CREAT | O_TRUNC,  0644);
if (fd != -1){
        printf("open fail\n");
}
```

read()读取操作

- 读取操作最常使用的是read()系统调用

```
#include <unistd.h>

ssize_t  read  (int fd,  void  *buf,   size_t  len);
```

- 从文件描述符fd所引用文件读取len个字节到buf
- read的返回值大小等于所读取字节的数目，如果发生错误等于-1
- 当read的返回值小于len的非零整数，可能的原因是可供读取的字节数目小于len
- read 返回值为0时指示达到了文件末端(end-of-file 简写 EOF)

write()写入系统调用

- 写入操作最常使用的是write()系统调用

```
#include <unistd.h>

ssize_t write (int fd, const void *buf, size_t count);
```

- 使用write()系统调用，会从buf开始将count个字节写入fd所指定的文件
- 附加模式
  - 当使用O_APPEND标志打开文件时，会在文件末端进行写入

lseek()系统调用

- 对文件读取和写入操作会导致文件位置变更。通过lseek()系统调用可以将文件位置设定为指定的值。除了改变文件位置，不会执行任何其他操作。

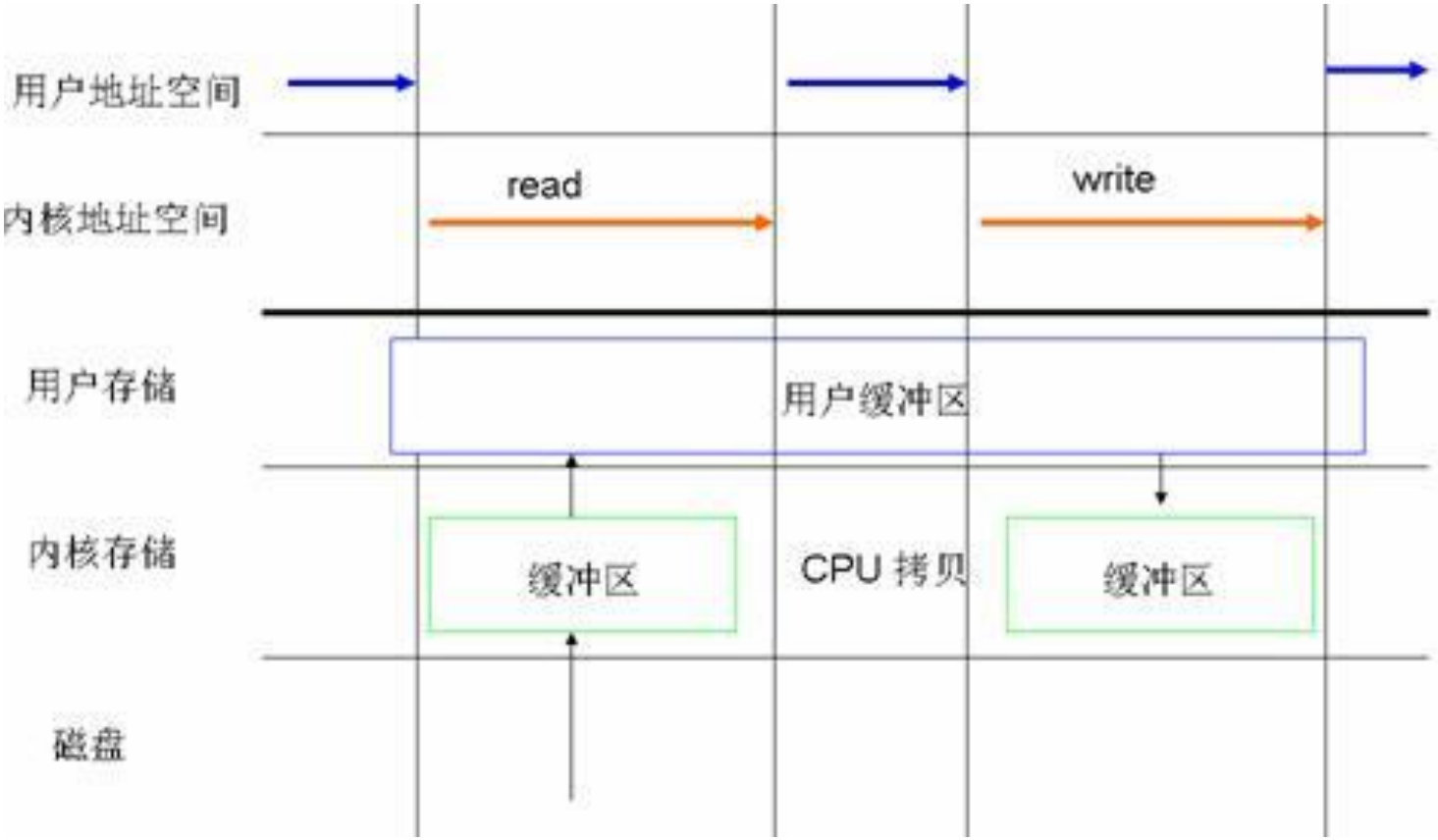
```
#include <sys/types.h>
#include <unistd.h>

off_t  lseek  (int fd,  off_t  pos,  int  origin);
```

- 调用成功时返回新的文件位置，错误时返回-1

origin参数

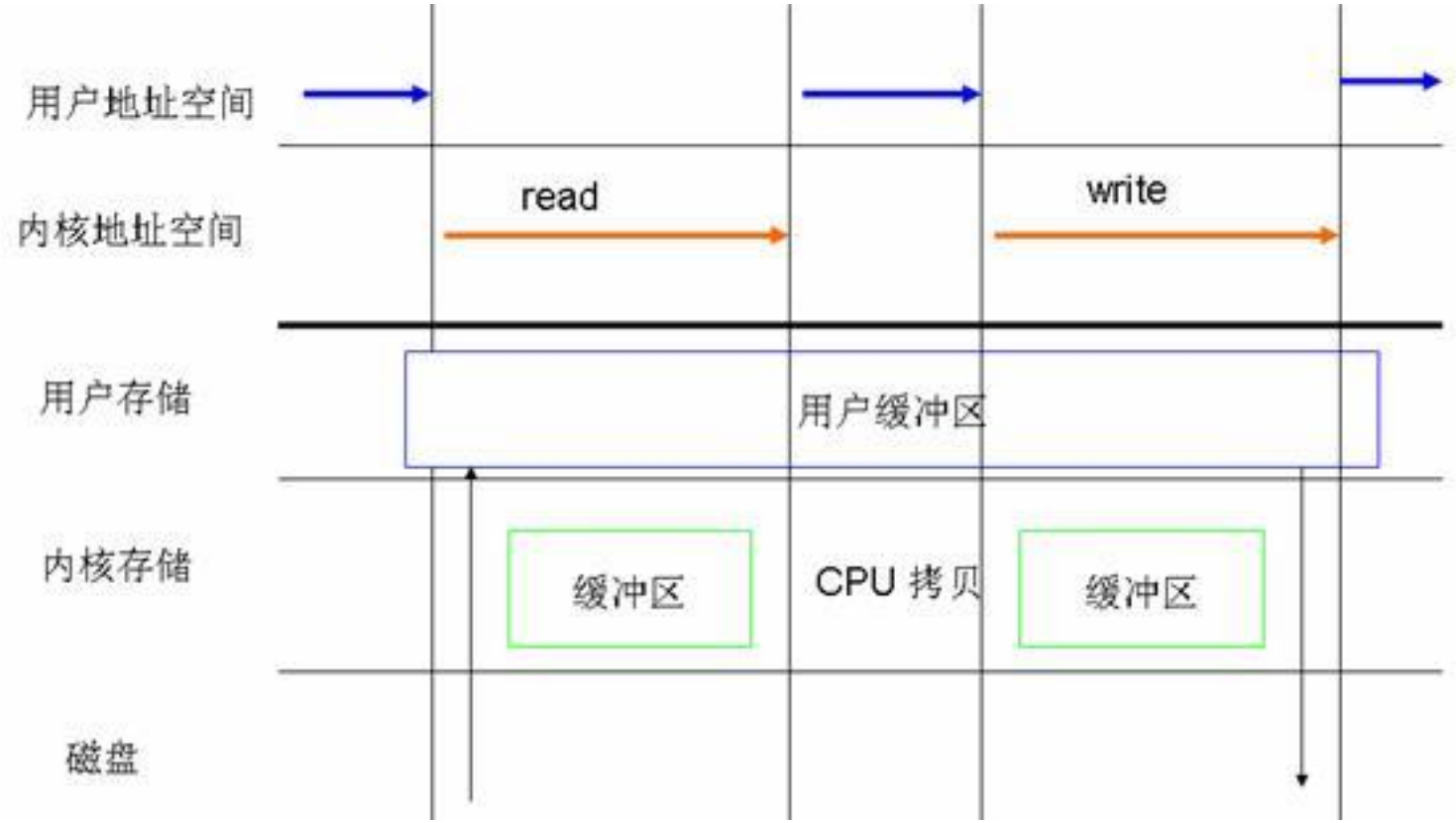
- lseek()调用的行为依赖于origin参数,该参数可以是以下任意值之一
- SEEK_CUR
  - 将文件位置当前值再加上pos个偏移值，pos可以是负值、0或正值。
- SEEK_END
  - 将文件位置设置成文件长度再加上pos个偏移值，pos可以是负值、0或正值。
- SEEK_SET
  - 将文件位置设置成pos值。如果pos值为0，就设置成文件开始。

标准I/O



以标准的方式对文件进行读写

直接I/O



数据传输不经过操作系统内核缓冲区

https://www.ibm.com/developerworks/cn/linux/l-cn-directio/index.html

close()系统调用

- 当应用程序使用完文件后，可以使用close()系统调用取消相关文件与文件描述符的映射关系

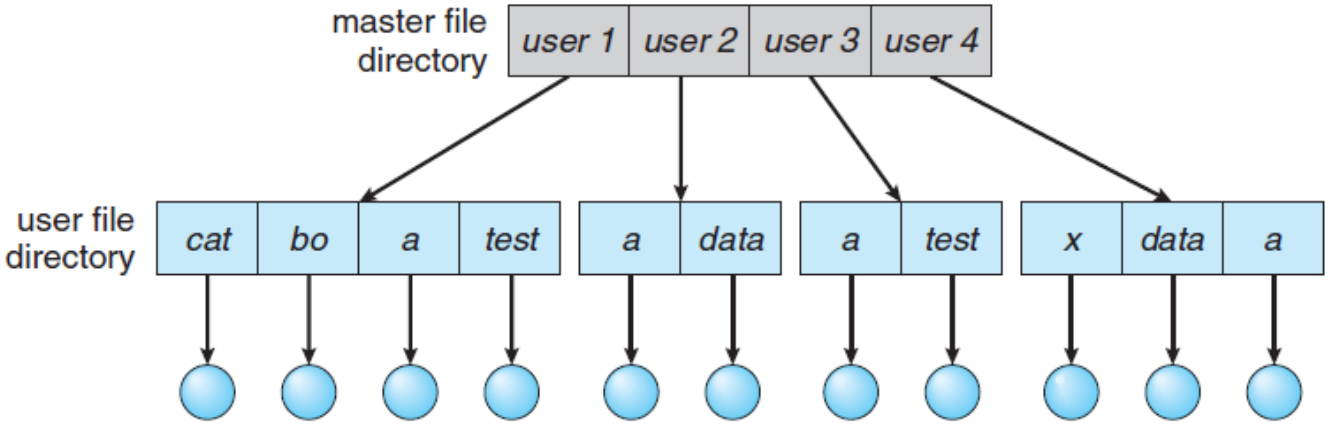
```
#include <unistd.h>

int  close  (int fd);
```

实验作业

- 本次实验模拟了实现了一个简单的文件系统。通过该模拟文件系统，了解文件系统的基本功能和实现框架。
- 本系统初始化10个用户，每个用户初始化5个文件，最多可拥有10个文件，所以每个用户可以在此基础上再创建5个文件，或删除后再创建
- 可以使用create、open、read、write、close、delete、dir来创建、打开、读、写、关闭、删除、和显示文件

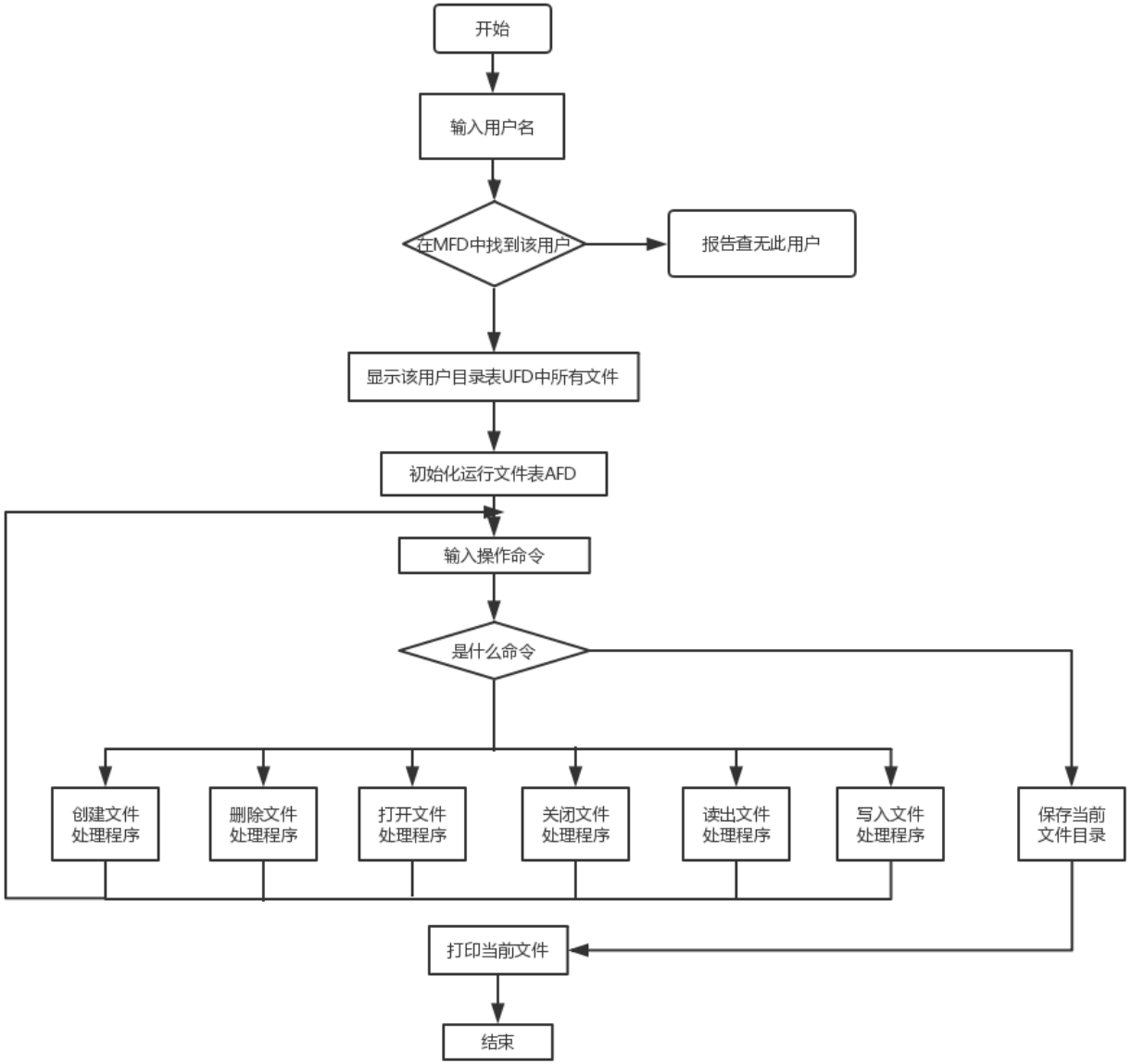
Two-Level Directory

- In the two-level directory structure, each user has his own **user file directory (UFD)**. When a user job starts or a user logs in, the system's **master file directory (MFD)** is searched.



Two-level directory structure.

模拟文件系统主要流程图



程序涉及的数据结构

1） 主目录结构 : 用于管理所有用户和用户目录

```
struct mdf {
        char uname[10];   //用户名
        UF    Udir;             //用户文件目录
} UFD[UserNumber];    //用户
```

2）用户文件目录结构体：用于管理用户的文件及文件权限

```
typedef struct {
        char fname[10];  //用户文件名
        int flag;               //文件存在标志
        int fprotect[3];    //文件保护码
        int flength;
} ;
```

程序涉及的数据结构

1） 主目录结构 : 用于管理所有用户和用户目录

```
struct afd
{
    char   opname[10];      //打开文件名
    int    flag;
    char   opfprotect[3];   //打开保护码
    int    rwpoint;         //读写指针
};
```

Lab Requirement

- Read the code, complete the report
- Package should be named as: OS_lab10_Name_xxxxxxxx where xxxxxxxx is your student id, Name is your name. This package should contain: your report.
- Check blackboard for ddl.

Thanks