# 文件描述符

## 简介

注：库函数即标准IO所在的位置。

在库函数层面表示的标准I/O用流来表示，在系统调用层面用文件描述字/符表示。不论是流还是文件描述字，表示的都是进程中与硬盘中文件的连接关系。

文件描述字：

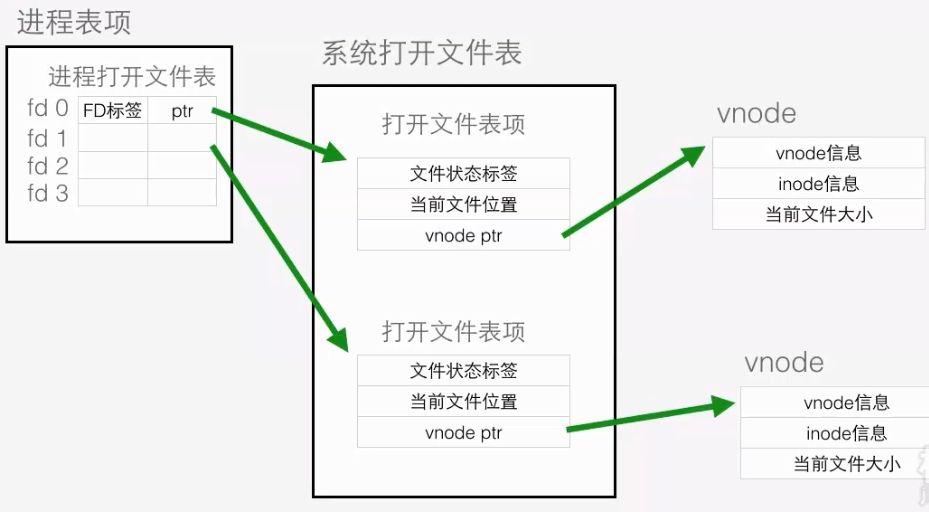
另一种连接机制（程序与文件）

系统调用（底层基本的IO操作）

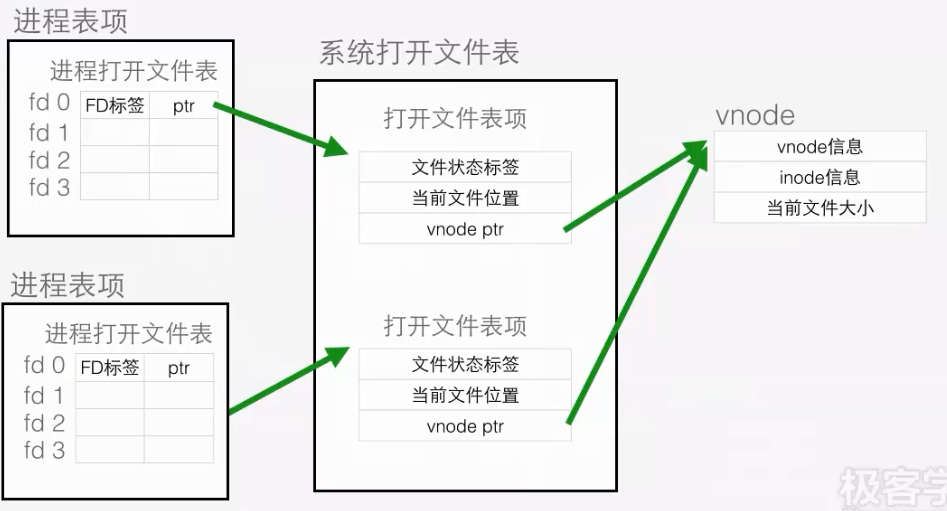
特定设备（流不行）

特殊方式进行IO（流不行）

## 内核数据结构

 一个进程打开多个文件：

进程表项是用户空间伴随着open创建的，系统打开文件表对应系统内核管理的数据结构。这样可以理解：一个进程中打开不同文件，进程里的文件不存在相互干扰。

 多个进程打开同一文件：

## 操作

### 创建文件

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

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

{

int ret = -1;

//创建新文件b.c，权限读写执行都有

ret = creat("b.c",0777);

if( -1 == ret )

{

perror("create");

return -1;

}

return 0;

}

### 打开文件

原型：

int open(const char \*pathname, int flags);//只是执行打开

int open(const char \*pathname, int flags, mode\_t mode);//还可以创建

数据类型（返回值）：int（范围有限制）

预先打开的文件描述字：0,1,2

示例（打开文件）：

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

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

{

int ret = -1;

//打开文件b.c

ret = open("b.c",O\_RDWR);//流 r+

//ret = open("b.c",O\_RDWR|O\_TRUNC);//流 w+

//ret = open("b.c",O\_RDWR|O\_APPEND);//流 a+

if( -1 == ret )

{

perror("create");

return -1;

}

close(ret);

return 0;

}

示例（打开并创建）：

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

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

{

int ret = -1;

//创建新文件b.c，权限读写执行都有,并且打开文件

ret = open("b.c",O\_RDWR|O\_CREAT,0777);

if( -1 == ret )

{

perror("create");

return -1;

}

close(ret);

return 0;

}

示例（文件存在正常打开，不存在则创建）：

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

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

{

int ret = -1;

//文件存在打开,不存在就创建打开

//创建新文件b.c，权限读写执行都有,并且打开文件

ret = open("b.c",O\_RDWR|O\_CREAT|O\_EXCL,0777);

if( -1 == ret )

{

if ( EEXIST == errno )//是否存在

{

ret = open("b.c",O\_RDWR);

if ( -1 == ret )

{

perror("open");

return -1;

}

}

return -1;

}

close(ret);

return 0;

}

示例（文件描述字）：

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

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

{

int ret = -1;

//打开文件b.c

while(1)

{

ret = open("b.c",O\_RDWR);//流 r+

if( -1 == ret )

{

perror("create");

return -1;

}

//显示文件描述字

printf("%d\n",ret);

}

close(ret);

return 0;

}

执行发现只能到1023，这个是可以突破上限的，通过ulimit –a查看open files，可以通过ulimit –n num修改

### 关闭文件

### 读写

#### read

示例：

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

#define MAX (256\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2)

//内存开的越大，耗时先减少后增加，这说明代码中使用用户分配的内存时，不是越大越好，也不是越小越好，应该根据系统的特性检测出一个比较正确的值

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

{

//拷贝test.mp4 到 new.mp4中

//打开文件，源 目的

int fd\_s = -1;//源文件 描述字

int fd\_d = -1;//目标文件 描述字

int ret = -1;

fd\_s = open("test.mp4",O\_RDONLY);//流 r 打开源文件

if (-1 == fd\_s )

{

perror("open");

return -1;

}

fd\_d = open("new.mp4",O\_WRONLY|O\_TRUNC|O\_CREAT,0777);//流 w 打开目标文件

if (-1 == fd\_d )

{

perror("open");

return -1;

}

//分配内存

char \*mem = NULL;

mem = malloc(MAX);

if ( NULL == mem )

{

perror("malloc");

return -1;

}

memset(mem,'\0',MAX);

//(读取－》写入)

while(1)

{

ret = read(fd\_s,mem,MAX);

if( -1 == ret )

{

perror("read");

return -1;

}

if( 0 == ret )

{

break;

}

ret = write(fd\_d,mem,ret);

if( -1 == ret )

{

perror("write");

return -1;

}

}

//释放内存

free(mem);

//关闭文件

close(fd\_s);

close(fd\_d);

return 0;

}

可以使用time ./a.out查看可执行文件的实际耗时，系统时间和用户时间。

#### write

#### 文件结束

#### IO效率

### 定位

示例：

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <fcntl.h>

#include <stdlib.h>

#include <string.h>

#define MAX (256\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2\*2)

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

{

//拷贝test.mp4 到 new.mp4中

//打开文件，源 目的

int fd\_s = -1;//源文件 描述字

int fd\_d = -1;//目标文件 描述字

int ret = -1;

fd\_s = open("test.mp4",O\_RDONLY);//流 r 打开源文件

if (-1 == fd\_s )

{

perror("open");

return -1;

}

fd\_d = open("new.mp4",O\_WRONLY|O\_TRUNC|O\_CREAT,0777);//流 w 打开目标文件

if (-1 == fd\_d )

{

perror("open");

return -1;

}

//计算出源文件大小

// 源文件定位到尾部， 计算出大小

long file\_size = -1;

file\_size = lseek(fd\_s,0,SEEK\_END);

if( -1 == file\_size )

{

perror("lseek");

return -1;

}

// 源文件定位到头部

ret = lseek(fd\_s,0,SEEK\_SET);

if( -1 == ret )

{

perror("lseek");

return -1;

}

//分配内存

char \*mem = NULL;

mem = malloc(file\_size);

if ( NULL == mem )

{

perror("malloc");

return -1;

}

memset(mem,'\0',file\_size);

//(读取－》写入)

//while(1)

// {

ret = read(fd\_s,mem,file\_size);

if( -1 == ret )

{

perror("read");

return -1;

}

if( 0 == ret )

{

// break;

}

ret = write(fd\_d,mem,ret);

if( -1 == ret )

{

perror("write");

return -1;

}

// }

//释放内存

free(mem);

//关闭文件

close(fd\_s);

close(fd\_d);

return 0;

}

### 重复

#### 重复描述字

其实复制的是文件描述字进程表项中的fd和fd标签，在系统层面的表项和属性全部都是一样的。

#### dup系列函数

原型：

int dup(int oldfd);

int dup2(int oldfd, int newfd);

示例：

#include <stdio.h>

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

{

int ret = -1;

int fd = -1;

fd = dup(1);//1 --- > 标准输出

if ( -1 == fd )

{

perror("dup");

return -1;

}

//写入fd － 》 1（标准输出）－》显示

ret = write(fd,"hello",5);

if ( -1 == ret )

{

perror("write");

return -1;

}

close(fd);

return 0;

}

### 与流互换

### 文件控制

### 缓冲

# 非阻塞IO