open的选项

2023年4月20日 9:29

O_RDONLY	以只读的方式打开
O_WRONLY	以只写的方式打开
O_RDWR	以读写的方式打开
O_CREAT	如果文件不存在，则创建文件

```c
int main(int argc, char *argv[])
{
    // ./0_open file1
    ARGS_CHECK(argc,2);
    // 使用要求，如果flags里面存在O_CREAT，必须选择3参数的open
    // 创建出来的文件权限会受到掩码的影响
    //int fd = open(argv[1],O_WRONLY|O_CREAT|O_EXCL,0666);
    int fd = open(argv[1],O_WRONLY|O_TRUNC);
    ERROR_CHECK(fd,-1,"open");
    printf("fd = %d\n", fd);
    close(fd);
    return 0;
}
```

fopen底层使用了open

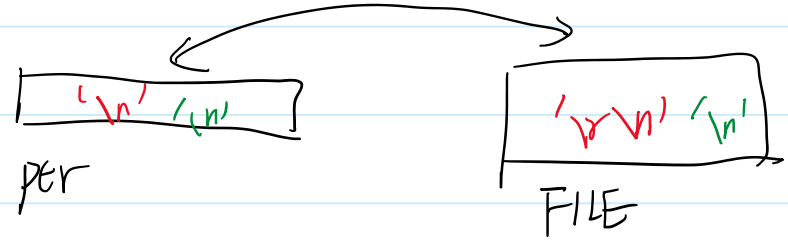
2023年4月20日 10:05

fopen() mode	open() flags
r	O_RDONLY
w	O_WRONLY | O_CREAT | O_TRUNC
a	O_WRONLY | O_CREAT | O_APPEND
r+	O_RDWR
w+	O_RDWR | O_CREAT | O_TRUNC
a+	O_RDWR | O_CREAT | O_APPEND

r 和 rb

只有在windows有区别

windows r rb



文本文件和二进制文件

2023年4月20日 10:09

文本文件 可以用 vim / cat 查看. 本质是一串字符序列 ASCII → 直接加载内存 字符数组.

```
file1+
  1 00000000: 3130 3030 3030 300a                    1000000.
~
```

二进制文件. 读写所用数据类型一致即可.

```
file1+
  1 00000000: 4042 0f00
```

```c
int main(int argc, char *argv[])
{
    // ./1_fopen_text_binary file1
    ARGS_CHECK(argc,2);
    FILE *fp = fopen(argv[1],"r+");
    ERROR_CHECK(fp,NULL,"fopen");
    //char buf[] = "1000000";
    //fwrite(buf,1,strlen(buf),fp);
    int i = 1000000;
    fwrite(&i,sizeof(int),1,fp);
    fclose(fp);
    return 0;

}
```

fscanf. 文本文件 → 内存中二进制数据

文件对象的读写 read和write

2023年4月20日 10:25

读取的上限.

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

long.

buf

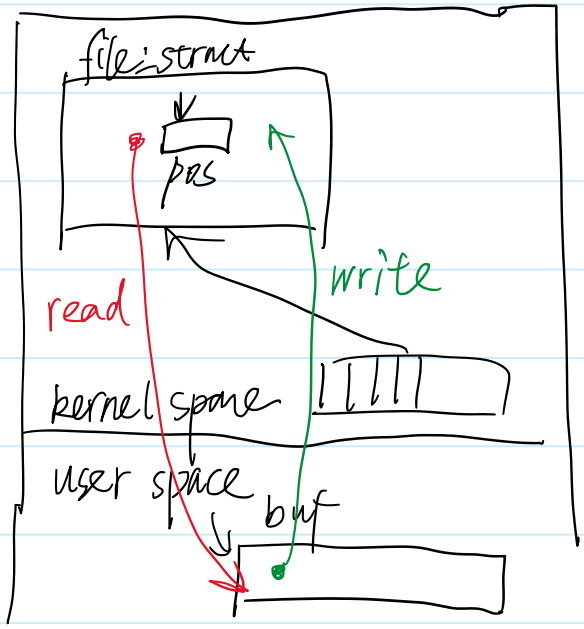
```
ssize_t write(int fd, const void *buf, size_t count);
```

buf

一般是实际写入的数量.

read的返回值.

- $0 < ret <= count$ 读取了ret
- $0 = ret$ 读到EOF.
- -1. 报错.



write的使用

2023年4月20日 11:02

```c
int main(int argc, char *argv[])
{
    // ./2_write file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    char buf[] = "how are you";
    ssize_t sret = write(fd,buf,strlen(buf));
    //ssize_t sret = write(fd,buf,sizeof(buf));
    //写入文本数据，应该使用strlen而不是sizeof
    ERROR_CHECK(sret,-1,"write");
    printf("sret = %ld\n", sret);
    close(fd);
    return 0;
}
```

read的故事

2023年4月20日 11:11

```c
int main(int argc, char *argv[])
{
    // ./3_read file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    char buf[6] = {0};
    ssize_t sret = read(fd,buf,5);
    ERROR_CHECK(sret,-1,"read");
    printf("buf = %s, sret = %ld\n",buf,sret);
    memset(buf,0,sizeof(buf));//每次read操作之前先清空
    sret = read(fd,buf,5);
    ERROR_CHECK(sret,-1,"read");
    printf("buf = %s, sret = %ld\n",buf,sret);
    close(fd);
    return 0;
}
```

read一个磁盘文件

read(fd, buf, count)

①读到字节 <= count

②每次read, pos后移.

③如果没有剩余内容. 继续read不会阻塞

会立刻返回0.

read读取标准输入 设备文件

2023年4月20日 11:21

4_read_stdin.c

```c
#include <49func.h>
int main()
{
    // 多申请一点内存
    char buf[1024] = {0};
    ssize_t sret = read(0,buf,sizeof(buf));
    ERROR_CHECK(sret,-1,"read");
    printf("sret = %ld, buf = %s\n",sret,buf);
    return 0;
}
```

read 设备文件 会将缓冲区内无数据 进程阻塞

读写二进制文件

2023年4月20日 11:34

5_write_binary.c buffers

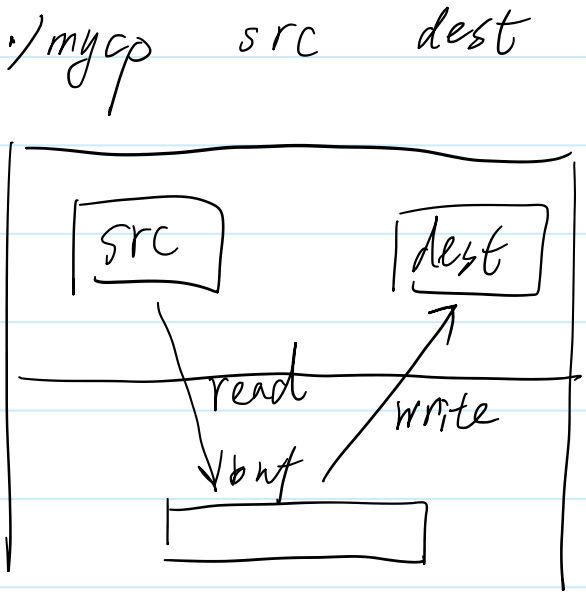
```c
#include <48func.h>
int main(int argc, char *argv[])
{
    // ./5_write_binary file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    int data = 100000;
    ssize_t sret = write(fd,&data,sizeof(int))
    ERROR_CHECK(sret,-1,"write");
    printf("sret = %ld\n", sret);
    close(fd);
    return 0;
}

~
```

5_read_binary.c

```c
#include <48func.h>
int main(int argc, char *argv[])
{
    // ./5_read_binary file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    int data;
    ssize_t sret = read(fd,&data,sizeof(int));
    ERROR_CHECK(sret,-1,"read");
    printf("sret = %ld\n", sret);
    ++data;
    printf("data = %d\n", data);
    close(fd);
    return 0;
}
```

实现一个cp命令

2023年4月20日 11:40



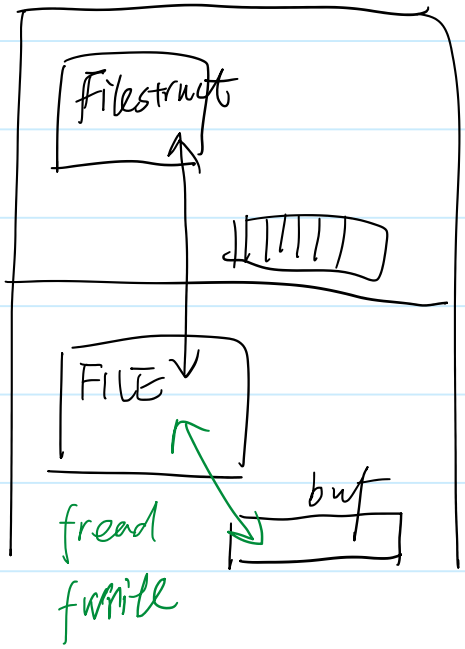

```c
int main(int argc, char *argv[])
{
    // ./6_cp src dest
    ARGS_CHECK(argc,3);
    int fdr = open(argv[1],O_RDONLY);
    ERROR_CHECK(fdr,-1,"open fdr");
    int fdw = open(argv[2],O_WRONLY|O_TRUNC|O_CREAT,0666);
    ERROR_CHECK(fdw,-1,"open fdw");
    //char buf[4096] = {0};
    char buf[4096000] = {0};
    // buf选择char数组，不是字符串的含义，而是因为char的字节是1
    while(1){
        memset(buf,0,sizeof(buf));
        ssize_t sret = read(fdr,buf,sizeof(buf));
        ERROR_CHECK(sret,-1,"read");
        // 读取磁盘文件，返回值为0，则读完
        if(sret == 0){
            break;
        }
        // 写入dest
        write(fdw,buf,sret);
    }
    close(fdr);
    close(fdw);
    return 0;
}
```

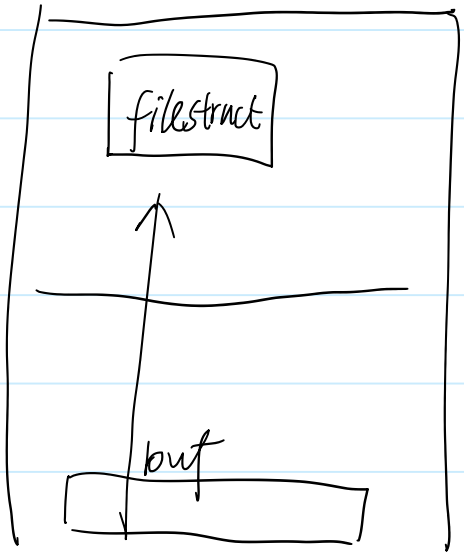
read次数越多 效率越差

read是系统调用，切换CPU状态。

fread/fwrite 对比 read/write

2023年4月20日 14:30





文件截断

2023年4月20日 14:38

```
int ftruncate(int fd, off_t length);
```

7_ftruncate.c

```c
#include <48func.h>
int main(int argc, char *argv[])
{
    // ./7_ftruncate file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    int ret = ftruncate(fd,40960);
    ERROR_CHECK(ret,-1,"ftruncate");
    close(fd);
    return 0;
}
```

stat命令

2023年4月20日 14:43

一个block是512B

```
-rw-rw-r-- 1 liao liao    99 Apr 20 09:33 Makefile
[liao@ubuntu Linuxday_07]$ stat file1
  File: file1
  Size: 10              Blocks: 8          IO Block: 4096   regular file
Device: 805h/2053d      Inode: 2248732     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/    liao)   Gid: ( 1000/    liao)
Access: 2023-04-20 14:43:06.425806386 +0800
Modify: 2023-04-20 14:43:06.425806386 +0800
Change: 2023-04-20 14:43:06.425806386 +0800
 Birth: -
```

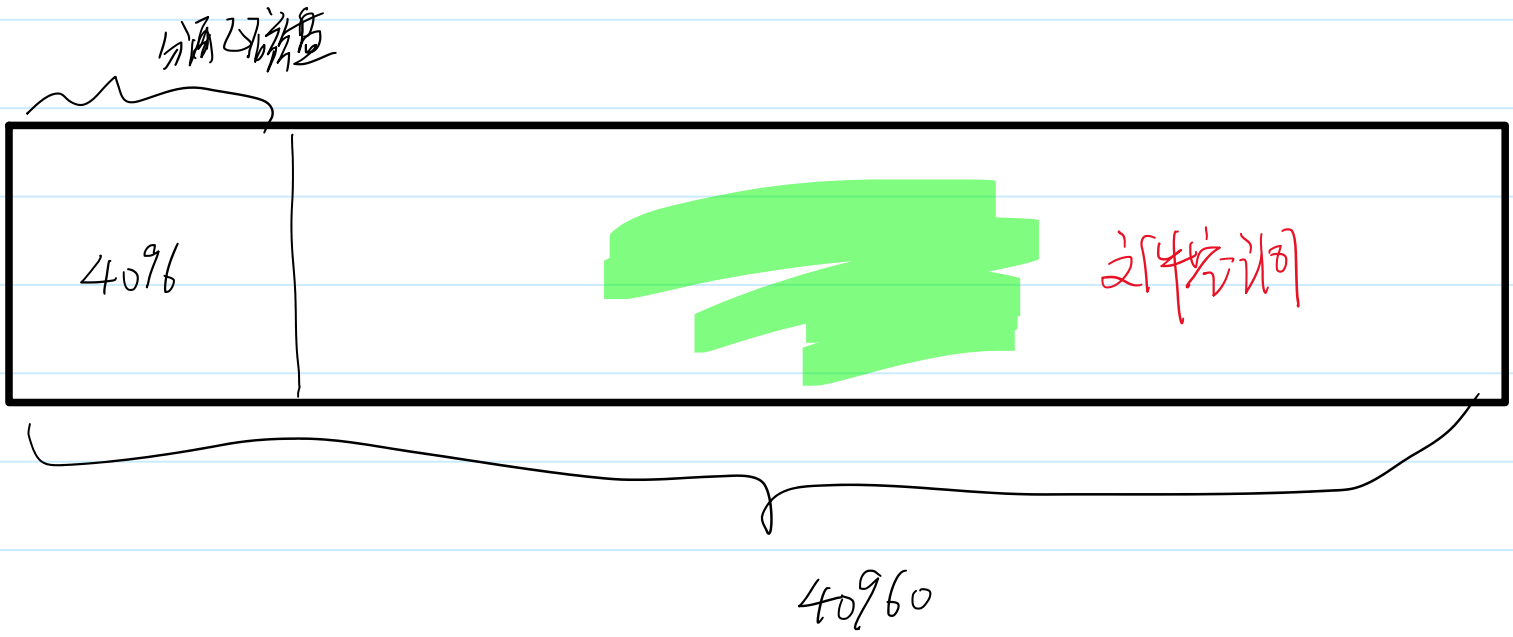
32x512 < 20312 < 40x512

```
[liao@ubuntu Linuxday_07]$ stat 0_open
  File: 0_open
  Size: 20312           Blocks: 40
```

文件空洞

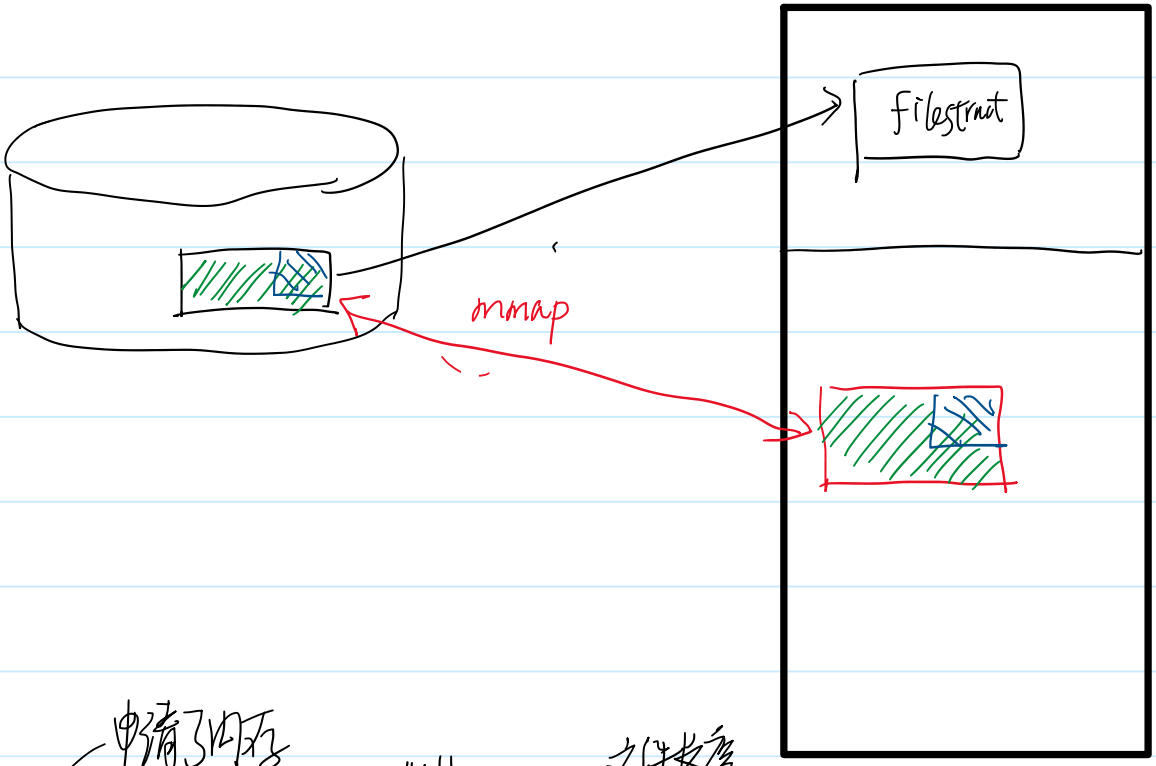
2023年4月20日 14:48

```
[liao@ubuntu Linuxday_07]$ stat file1
  File: file1
  Size: 40960           Blocks: 8
```



文件映射 mmap

2023年4月20日 14:52



申请3G内存

NULL

文件长度

MAP_SHARED

```
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);
```

释放

先open

0

长度.

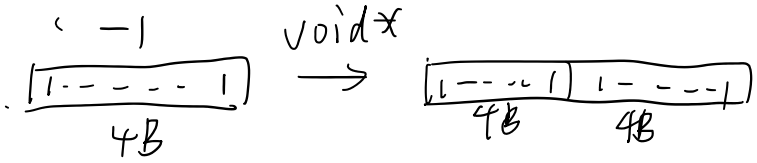
open(O_RDWR)

PROT_READ | PROT_WRITE

mmap的返回值

mmap的例子

2023年4月20日 15:12




```c
#include <49func.h>
int main(int argc, char *argv[])
{
    // ./8_mmap file1
    ARGS_CHECK(argc,2);
    // 先 open 文件
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    // 建立内存和磁盘之间的映射
    char *p = (char *)mmap(NULL,5,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
    ERROR_CHECK(p,MAP_FAILED,"mmap");//mmap失败返回不是NULL
    for(int i = 0; i < 5; ++i){
        printf("%c", *(p+i));
    }
    printf("\n");
    *(p+4) = 'O';
    munmap(p,5);
    close(fd);
    return 0;
}
```

lseek

2023年4月20日 15:22

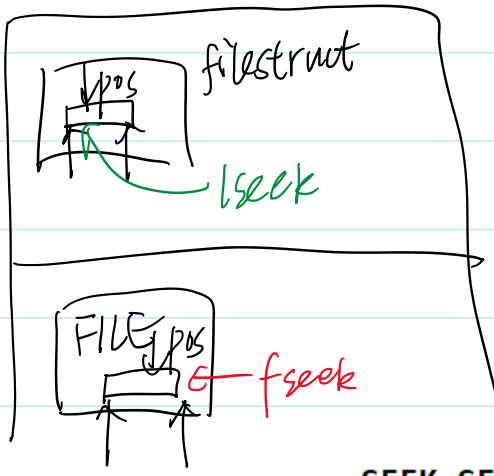
```
off_t lseek(int fd, off_t offset, int whence);
```

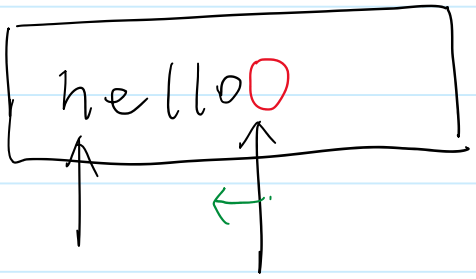
偏移offset

+ 向右

- 向左

以whence为起点





SEEK_SET

TH

SEEK_CUR

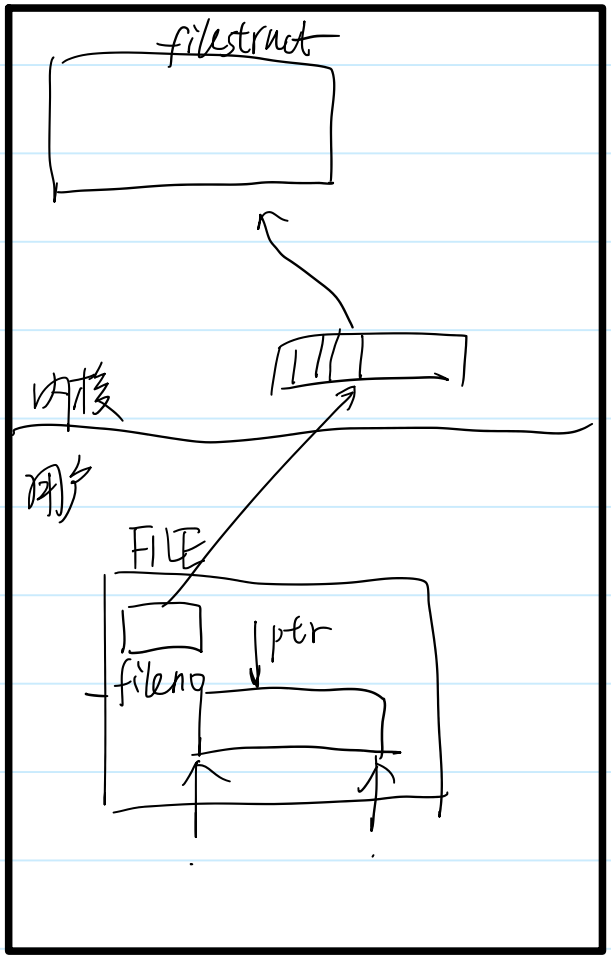
TH

SEEK_END

```c
int main(int argc, char *argv[])
{
    // ./9_lseek file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    write(fd,"hello",5);
    lseek(fd,-1,SEEK_CUR);
    write(fd,"O",1);
    close(fd);
    return 0;
}
```

FILE和文件对象

2023年4月20日 15:56




```c
#include <49func.h>
int main(int argc, char *argv[])
{
    // ./11_fileno file1
    ARGS_CHECK(argc,2);
    FILE * fp = fopen(argv[1],"r+");
    ERROR_CHECK(fp,NULL,"fopen");
    write(3,"hello",5);
    fclose(fp);
    return 0;
}
```

魔数

```c
struct _IO_FILE
{
  int _flags;
  char *_IO_read_ptr;
  char *_IO_read_end;
  char *_IO_read_base;
  char *_IO_write_base;
  char *_IO_write_ptr;
  char *_IO_write_end;
  char *_IO_buf_base;
  char *_IO_buf_end;
  char *_IO_save_base;
  char *_IO_backup_base;
  char *_IO_save_end;
  struct _IO_marker *_markers;
  struct _IO_FILE *_chain;
  int _fileno;
```

fileno

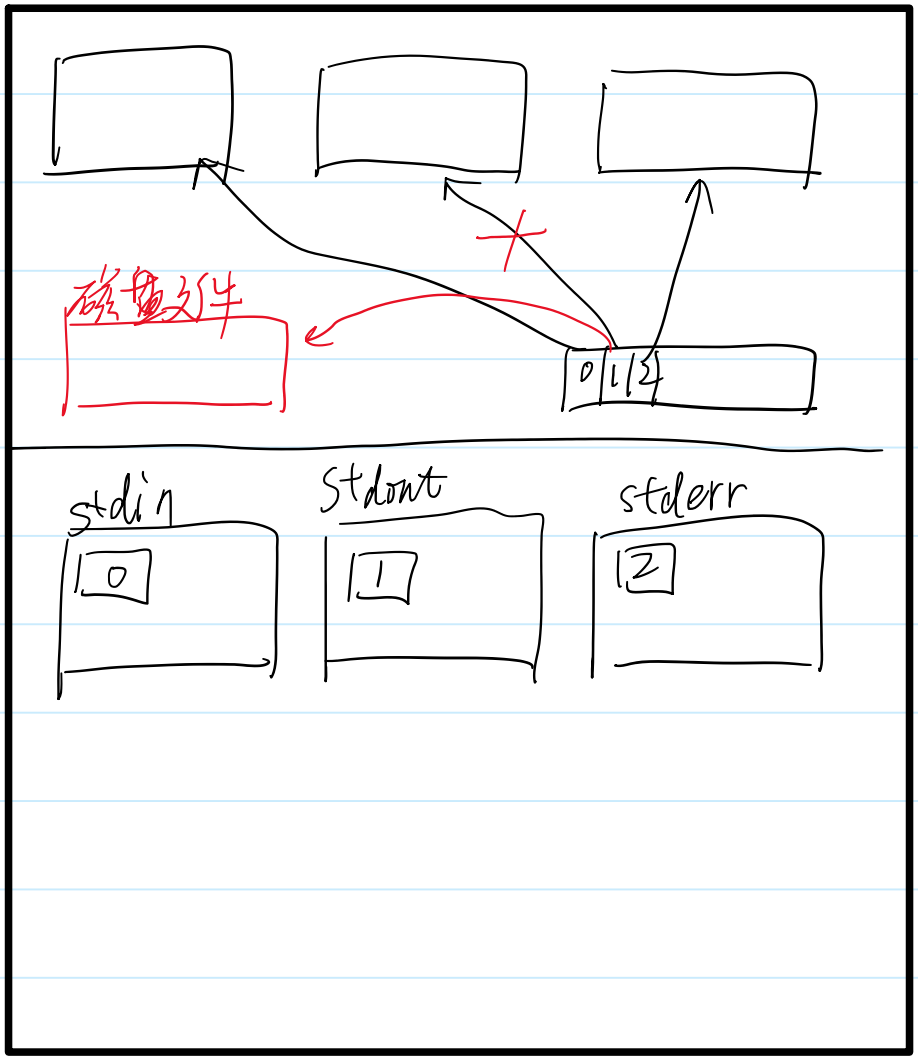
2023年4月20日 16:09

```c
int fileno(FILE *stream);

int main(int argc, char *argv[])
{
    // ./11_fileno file1
    ARGS_CHECK(argc,2);
    FILE * fp = fopen(argv[1],"r+");
    ERROR_CHECK(fp,NULL,"fopen");
    //write(3,"hello",5);
    //write(fp->_fileno,"world",5);
    write(fileno(fp),"hello",5);
    fclose(fp);
    return 0;
}
```

面向接口编程.

任何程序启动就会创建3个文件流

2023年4月20日 16:12



12_std.c

```c
#include <49func.h>
int main()
{
    printf("stdin fd = %d\n", fileno(stdin));
    printf("stdout fd = %d\n", fileno(stdout));
    printf("stderr fd = %d\n", fileno(stderr));
    return 0;
}
```

先close(1)，再open("file1")

file1的文件对象的fd=1

printf → stdout → write(1,...)

重定向

2023年4月20日 16:19

重定向stdout，更改1号fd引用的文件对象.

```c
int main(int argc, char *argv[])
{
    // ./13_redirect file1
    ARGS_CHECK(argc,2);
    // 在关闭1之前请先打印一个换行符
    printf("You can see me!\n");
    //close(1);
    close(STDOUT_FILENO);
    int fd = open(argv[1],O_WRONLY);
    ERROR_CHECK(fd,-1,"open");
    printf("fd = %d\n", fd);
    printf("You can't see me!\n");
    return 0;
}
```

```
[liao@ubuntu Linuxday_07]$ ./13_redirect file1
You can see me!
[liao@ubuntu Linuxday_07]$ cat file1
fd = 1
You can't see me!
```

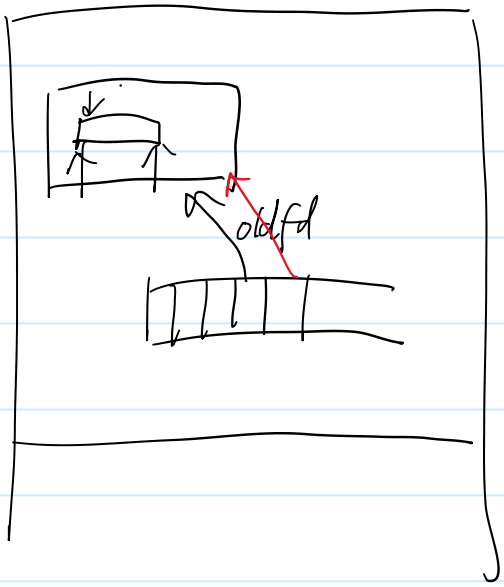
dup

2023年4月20日 16:25

```
int dup(int oldfd);
```

选择一个空的fd.

让新fd和oldfd引用同个文件对象.



文件对象采用引用计数机制管理

↑

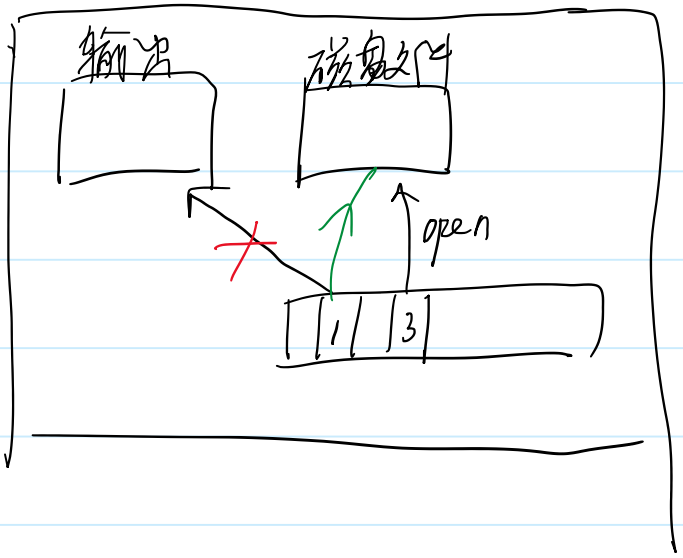
引用计数 → 0，才会真正释放

.close

```c
int main(int argc, char *argv[])
{
    // ./14_dup file1
    ARGS_CHECK(argc,2);
    int oldfd = open(argv[1],O_RDWR);
    ERROR_CHECK(oldfd,-1,"open");
    printf("oldfd = %d\n",oldfd);
    int newfd = dup(oldfd);
    ERROR_CHECK(newfd,-1,"dup");
    printf("newfd = %d\n", newfd);
    // 新旧文件描述符数值不相同
    // 引用的文件对象相同，共享偏移量
    write(oldfd,"hello",5);
    write(newfd,"world",5);
    close(newfd);
    close(oldfd);
    return 0;
}
```

先打开文件，再实现重定向

2023年4月20日 16:38



open.

close(1)

dup

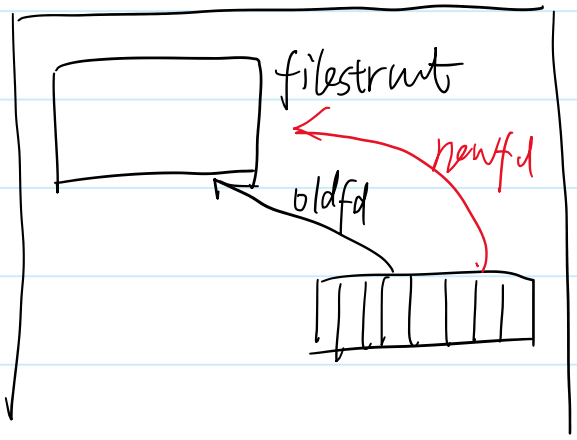
```c
#include <49func.h>
int main(int argc, char *argv[])
{
    // ./15_redirect file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_WRONLY);
    ERROR_CHECK(fd,-1,"open");
    printf("fd = %d\n", fd);
    printf("You can see me!\n");

    close(STDOUT_FILENO);
    dup(fd);//让1号文件描述符引用磁盘文件的文件对象
    printf("You can't see me!\n");
    close(fd);
    return 0;
}
```

dup2

2023年4月20日 16:49

```
int dup2(int oldfd, int newfd);
```

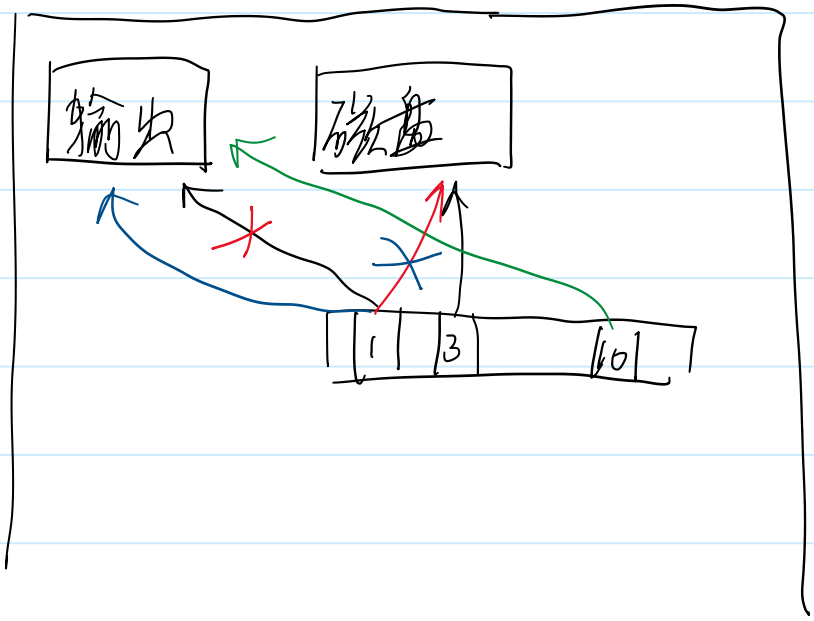
用户自行指定的newfd



如果dup2之前，newfd已引用一个文件对象，
dup2调用时，会自动close.

左右横跳

2023年4月20日 17:11



dup2(stdout, 10)

dup2(3 , stdout)

dup2(10, stdout)

代码

2023年4月20日 17:19

```c
#include <49func.h>
int main(int argc, char *argv[])
{
    // ./17_redirect file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_WRONLY);
    ERROR_CHECK(fd,-1,"open");
    printf("我过来啦！ \n");
    int newfd = 10;
    dup2(STDOUT_FILENO,newfd);//使用newfd备份输出文件对象
    //让1引用磁盘文件
    dup2(fd,STDOUT_FILENO);
    printf("我过去啦！ \n");
    //让1引用输出设备
    dup2(newfd,STDOUT_FILENO);
    printf("我又回来啦！ \n");
    close(fd);
    return 0;
}
```

数值永远是1

有名管道

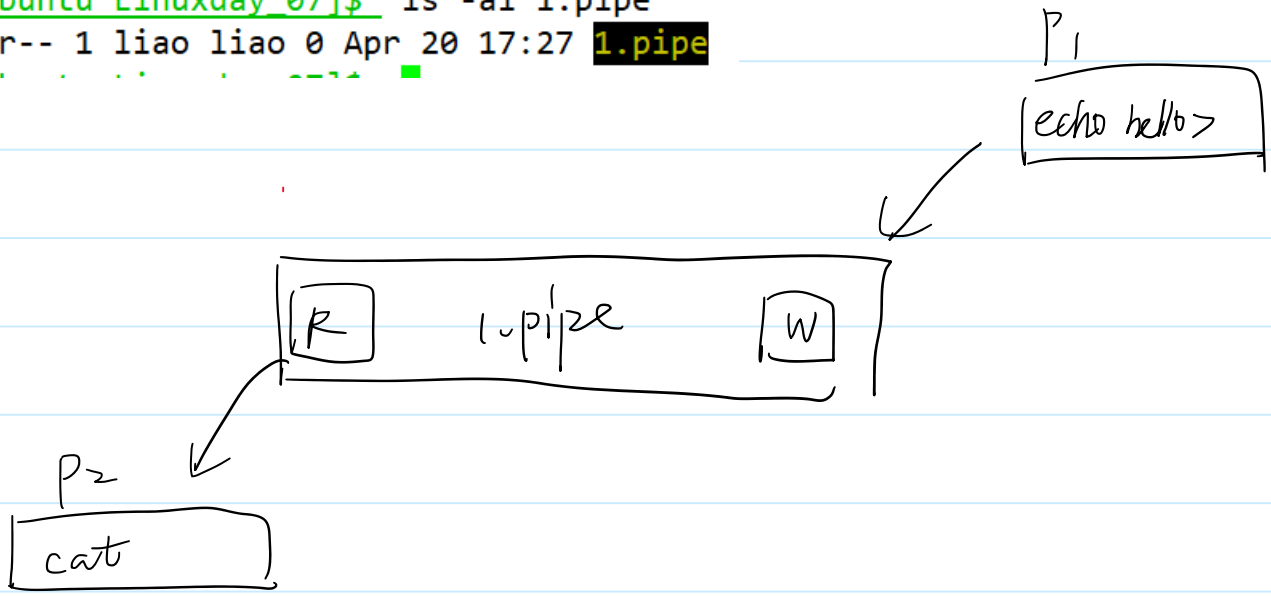
2023年4月20日 17:24

named pipe / FIFO

在文件系统存在路径

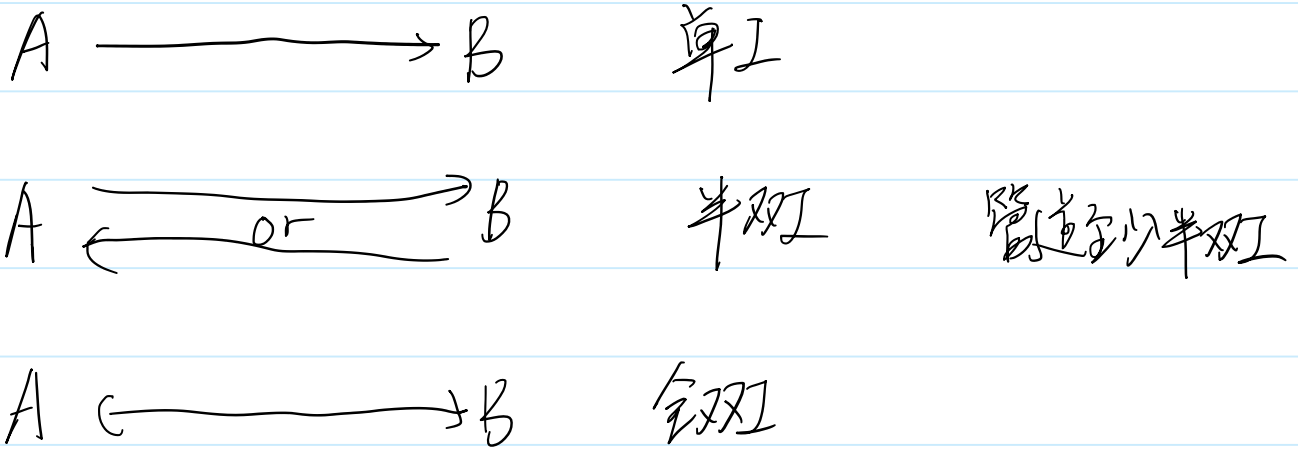
一种特殊的文件. 一种进程间通信进制在文件系统的映射

```
mkfifo - make FIFOs (named pipes)

[liao@ubuntu Linuxday_07]$ mkfifo 1.pipe
[liao@ubuntu Linuxday_07]$ ls -al 1.pipe
prw-rw-r-- 1 liao liao 0 Apr 20 17:27 1.pipe
```



通信类型

2023年4月20日 17:36



一般在使用上，用户会把一条管道当单工使用.

、非要全双工 启用两条管道.

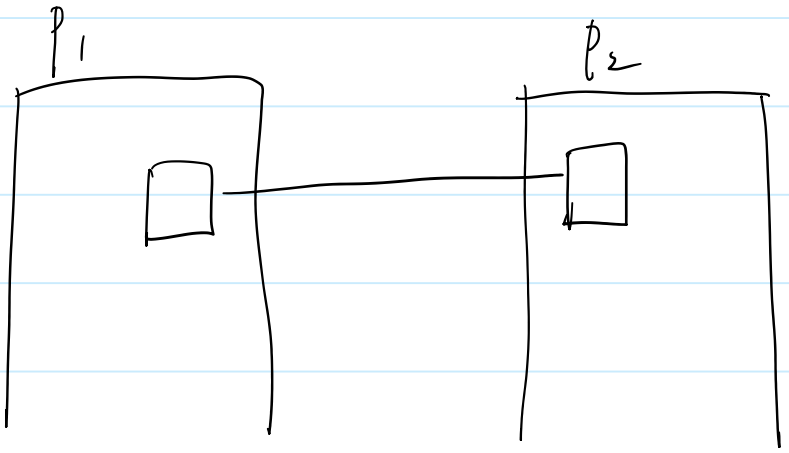
open close read write

2023年4月20日 17:40

open 打开管道的一端

O_RDONLY 读端

O_WRONLY 写端



open 会引发阻塞，等到对端也被open了，才会恢复就绪.

```c
// ./18_open_pipe_write 1.pipe
ARGS_CHECK(argc,2);
int fdw = open(argv[1],O_WRONLY);
ERROR_CHECK(fdw,-1,"open");
printf("write side is opened!\n");
close(fdw);
return 0;
```

```c
// ./18_open_pipe_read 1.pipe
ARGS_CHECK(argc,2);
int fdr = open(argv[1],O_RDONLY);
ERROR_CHECK(fdr,-1,"open");
printf("read side is opened!\n");
close(fdr);
return 0;
```