自我介绍

2023年8月4日 9:27

泥鳅

湖南常德

课堂要求

2023年8月4日 9:54

时间 9点 2点 7点半

课堂纪律

- 不要随意进出教室
- 不要戴耳机

课间要求

- 休息去公区
- 不要打游戏

学习方法

2023年8月4日 10:00

以老师为中心

禁止上课敲代码

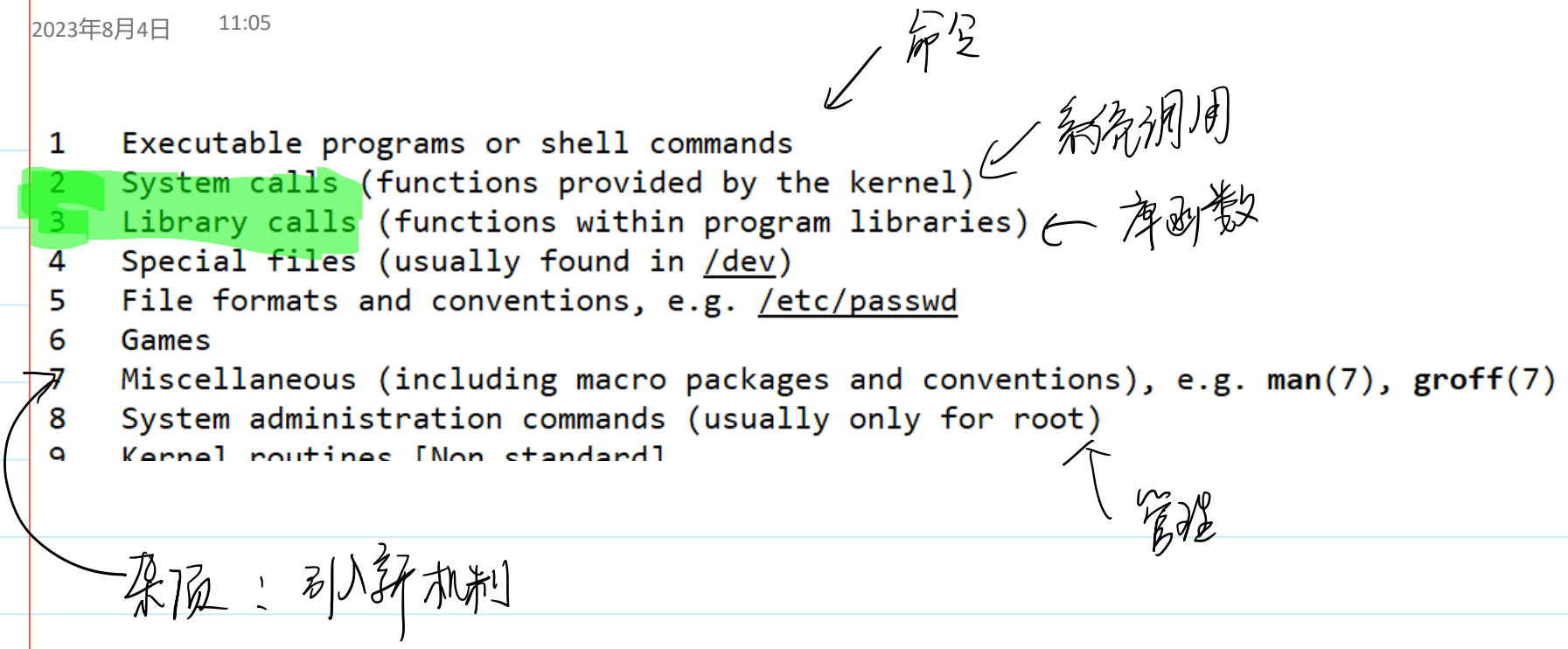
理解**模型**

课程大纲

2023年8月4日 10:12

阅读帮助手册

2023年8月4日 11:05




```
1   Executable programs or shell commands
2   System calls (functions provided by the kernel)
3   Library calls (functions within program libraries)
4   Special files (usually found in /dev)
5   File formats and conventions, e.g. /etc/passwd
6   Games
7   Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
8   System administration commands (usually only for root)
9   Kernel routines [Non standard]
```

阅读帮助手册

2023年8月4日 11:10

1. NAME.

2. 声明:

```
char *getcwd(char *buf, size_t size);
```

传入传出

返回值类型. int -1 报错 0 正常.

void. 不会报错.

指针类型 不可能返回局部变量

有可能分配在主调方栈帧上

有可能分配在堆上(其他区域)

"有副作用" → 释放

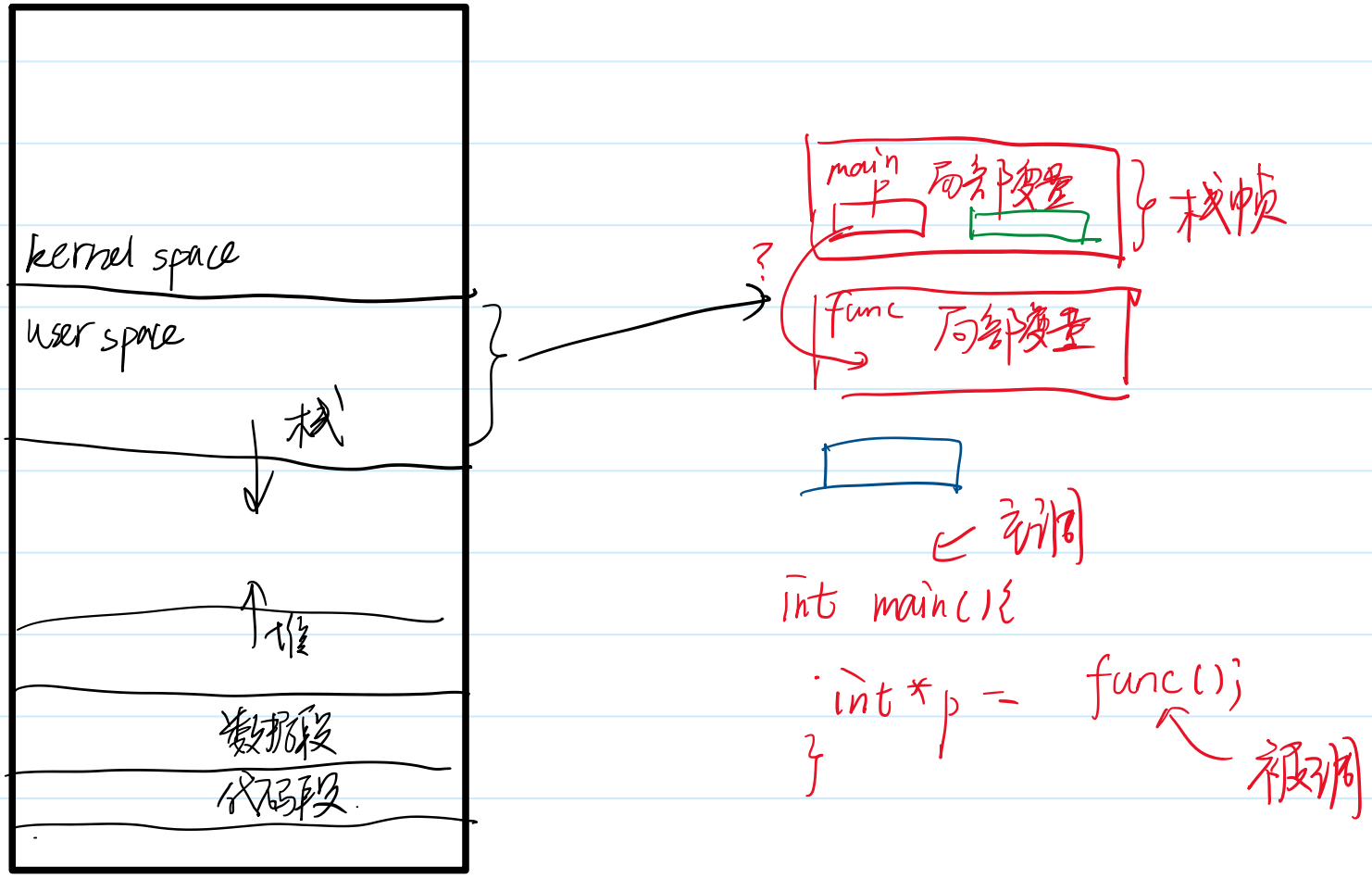
fopen → fclose

opendir → closedir

malloc → free

函数调用内存模型

2023年8月4日 11:17

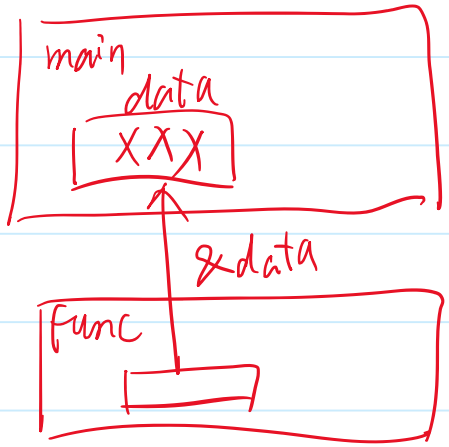


指针类型的参数

2023年8月4日 11:27

一、 被调修改主调的内存

不带const 传入传出参数

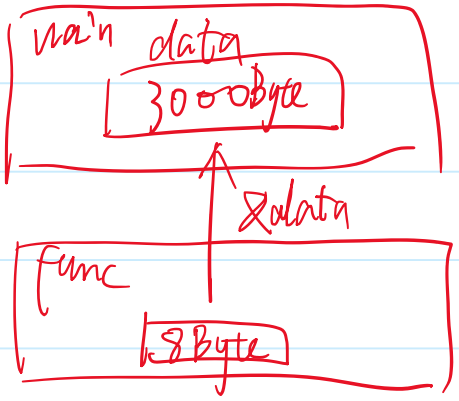


二、 data很大.

带const. 传入参数

data不会被

func修改.



指针参数使用问题

2023年8月4日 11:33

```
int func(int *arg);

int main(){
    int *p;
    func(p);
}
```

野指针

申请内存，取地址

```
int p; func(&p);
int p[3]; func(p);
int *p = malloc(4); func(p);
```

```
int func1(char *buf)
int func2(const char *buf)
```

```
int main(){
    func1("Hello");
    func2("Hello");
}
```

"Hello" 只读

阅读帮助手册

2023年8月4日 11:44

3. 返回值，出错的条件 → errno

→ perror

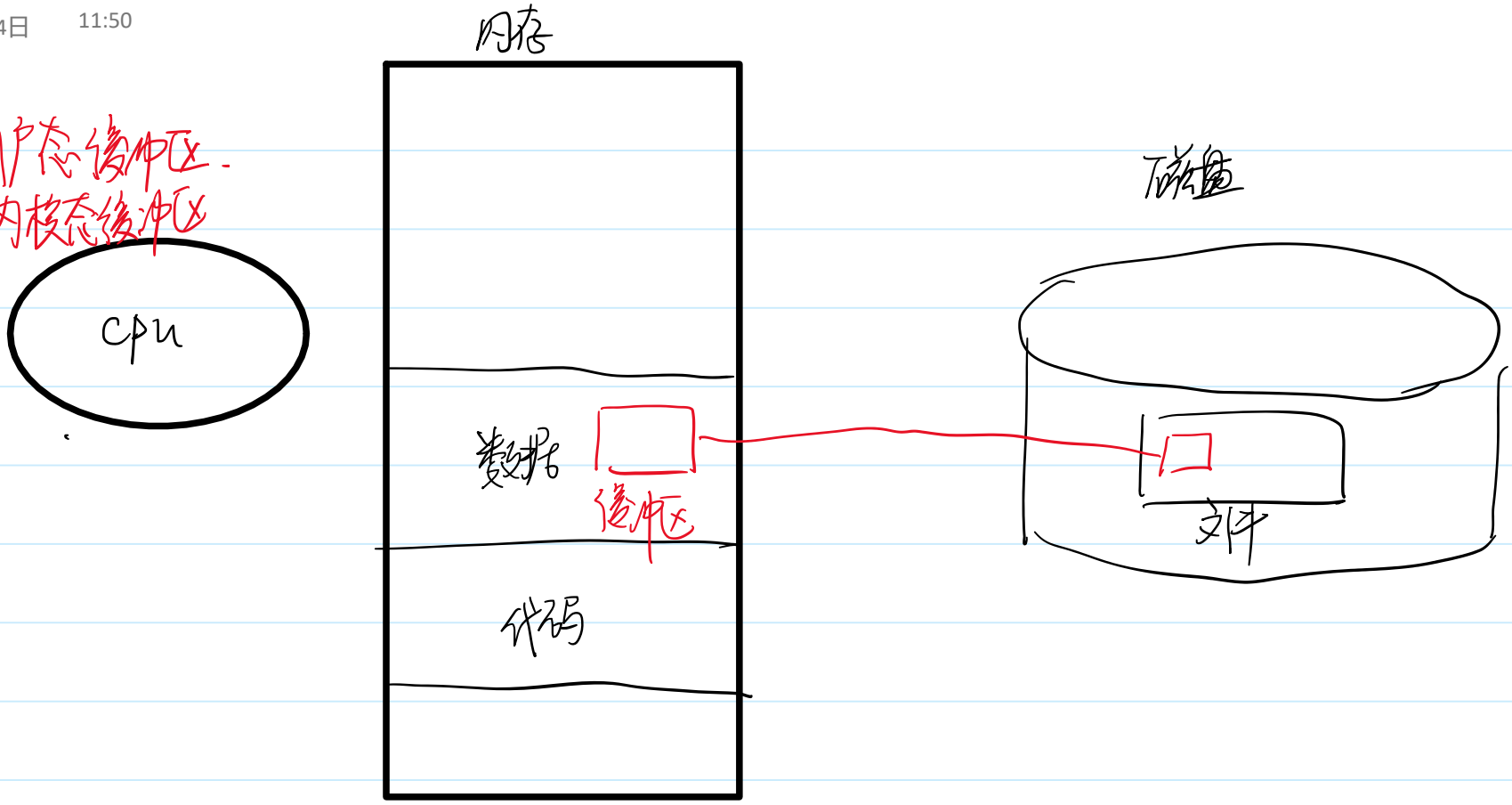
ERROR_CHECK

4. 描述 不要从头到尾 按需查看

无缓冲的文件IO

2023年8月4日 11:50

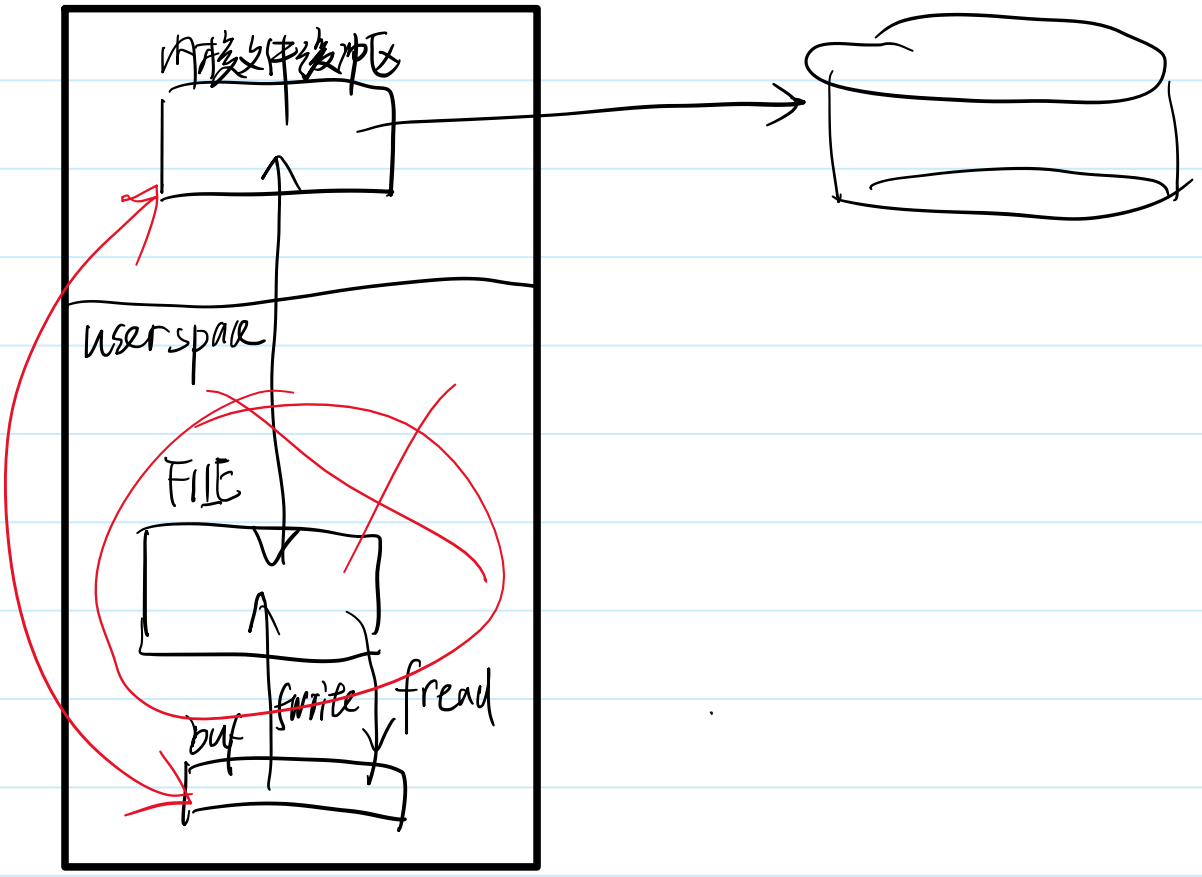
无用户态缓冲区
有内核态缓冲区



fopen

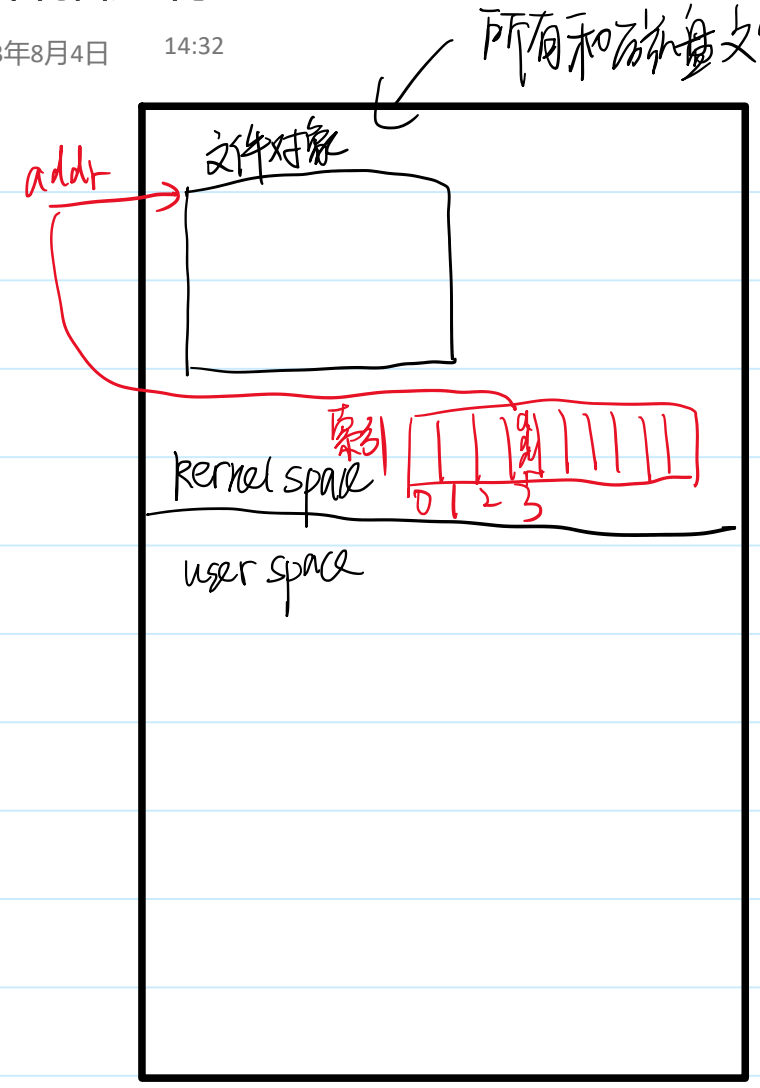
2023年8月4日 11:56

FILE 文件流/用户态文件缓冲区



文件描述符

2023年8月4日 14:32



cpu { 用户 只能执行部分指令
内核 执行所有指令

索引数组 → 下标：文件描述符
→ 元素：文件对象的地址

open close

2023年8月4日 14:43

路径 属性 新建文件的权限

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

32bit

只读、只写、读写

是否新建文件.

按位或.

int flag 看成32位的数据

每个属性独立取值,

同时拥有多个属性 按位或.

同种属性不能或.

只读不能或只写

```
#include <unistd.h>

int close(int fd);
```

2023年8月4日 14:58

三选一

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
O_EXCL	仅与O_CREAT连用，如果文件已存在，则open失败
O_TRUNC	如果文件存在，将文件的长度截至0
O_APPEND	已追加的方式打开文件，每次调用write时，文件指针自动先移到文件尾，用于多进程写同一个文件的情况。

文件描述符

2023年8月4日 15:01

return the new file descriptor, or -1 if an error occurred

0_open.c

```c
#include <52func.h>
int main(int argc, char *argv[])
{
    // ./0_open file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_WRONLY|O_CREAT,0666);
    // 新建文件还要受到掩码的影响
    ERROR_CHECK(fd,-1,"open");
    printf("fd = %d\n" ,fd);
    close(fd);
    return 0;
}
```

使用要求

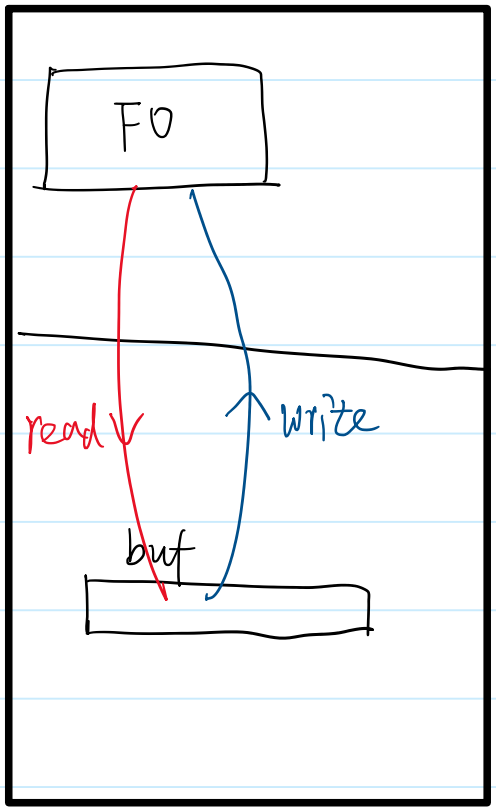
2023年8月4日 15:12

如果 open的flags 存在 O_CREAT，请使用3参数的版本

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

read和write

2023年8月4日 15:18



long

读取上限

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

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

实际写入的字节数.

```
char buf[100] = {0};
read(fd, buf, sizeof(buf) / strlen(buf))

char buf[100] = "hello";
write(fd, buf, sizeof(buf) / strlen(buf))
```

字符串

文本文件 二进制文件

2023年8月4日 15:50

```
:%!xxd
```

```
file2+
  1 00000000: 8096 9800 0a
```

```c
// 1_write.c
#include <51func.h>
int main(int argc, char *argv[])
{
    // ./1_write file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_WRONLY);
    ERROR_CHECK(fd,-1,"open");
    write(fd,"10000000",8);
    close(fd);
    return 0;
}
```

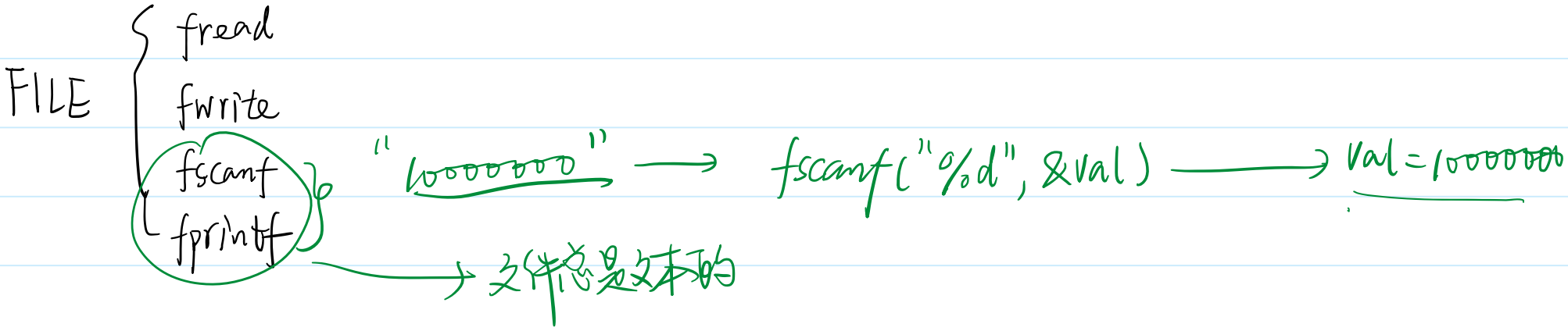
文本

```c
#include <51func.h>
int main(int argc, char *argv[])
{
    // ./1_write_binary file2
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_WRONLY);
    ERROR_CHECK(fd,-1,"open");
    int val = 10000000;
    write(fd,&val,sizeof(val));
    close(fd);
    return 0;
}
```

二进制

6个读写的函数 .

2023年8月4日 16:00



fd { read write }

文件是文本的，读到的字符串.

文件是二进制，怎么写就怎么读.

二进制的读操作

2023年8月4日 16:12

1_read_binary.c

```c
#include <52func.h>
int main(int argc, char *argv[])
{
    // ./1_read_binary file2
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    int val; // 怎么写 就 怎么读
    read(fd,&val,sizeof(val));
    printf("val = %d\n", val);
    close(fd);
    return 0;
}

```

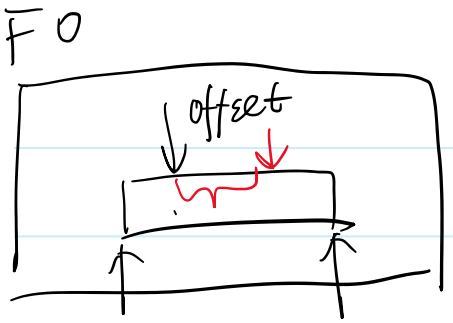
~

深入read

2023年8月4日 16:13

```c
int main(int argc, char *argv[])
{
    // ./2_read file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDONLY);
    ERROR_CHECK(fd,-1,"open");
    char buf[1024] = {0};
    ssize_t sret = read(fd,buf,4);
    printf("sret = %ld, buf = %s\n", sret, buf);
    memset(buf,0,sizeof(buf));//读取操作前记得清空
    sret = read(fd,buf,4);
    printf("sret = %ld, buf = %s\n", sret, buf);
    memset(buf,0,sizeof(buf));
    sret = read(fd,buf,4);
    printf("sret = %ld, buf = %s\n", sret, buf);
    close(fd);
    return 0;
}
```

磁盘文件.




```
liao:LinuxDay08$ ./2_read file1
sret = 4, buf = 1234
sret = 4, buf = 5678
sret = 0, buf =
```

① read之前记得清空buf

② 连续read,会修改FO的offset

③ 没有剩余内容.

read不会阻塞,会立刻返回0.

read 设备文件

2023年8月4日 16:27

任何进程一启动就打开3个文件

```
stderr -> /proc/self/fd/2
stdin -> /proc/self/fd/0
stdout -> /proc/self/fd/1
```

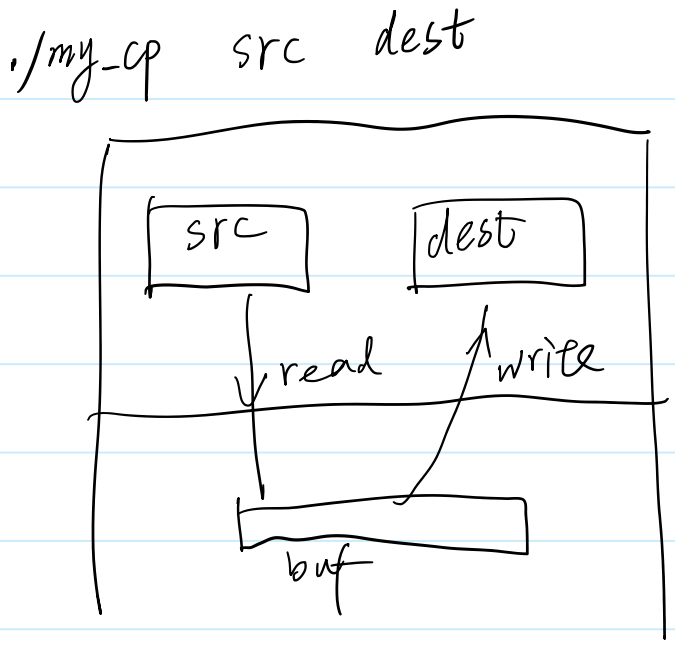
3_read_stdin.c

```c
#include <52func.h>
int main()
{
    // ./3_read_stdin
    char buf[1024] = {0};
    //read(0,buf,sizeof(buf));
    ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
    printf("sret = %ld, buf = %s\n", sret, buf);
    return 0;
}
```

read设备会引发阻塞

read write 实现一个cp命令

2023年8月4日 16:38

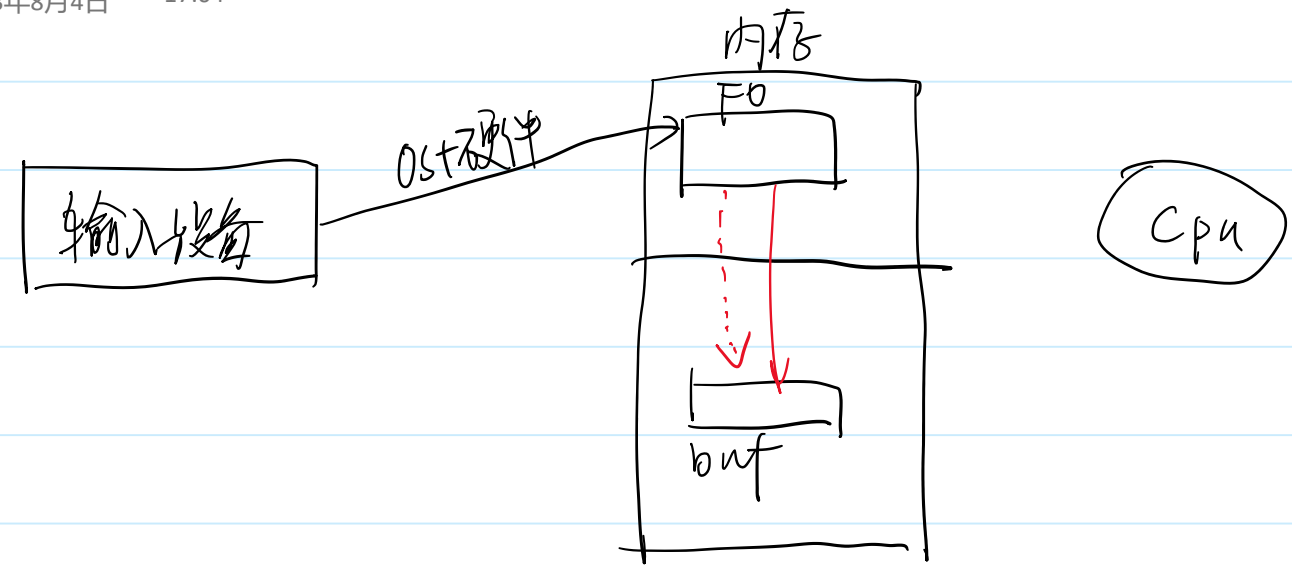



```c
int main(int argc, char *argv[])
{
    // ./4_mycp src dest
    ARGS_CHECK(argc,3);
    int fdr = open(argv[1],O_RDONLY);
    ERROR_CHECK(fdr,-1,"open fdr");
    int fdw = open(argv[2],O_WRONLY|O_CREAT,0666);
    ERROR_CHECK(fdw,-1,"open fdw");
    char buf[1024] = {0};
    while(1){
        memset(buf,0,sizeof(buf));
        ssize_t sret = read(fdr,buf,sizeof(buf));
        if(sret == 0){
            break;
        }
        write(fdw,buf,sret);
        // sizeof 固定是1024
        // strlen 如果文件是二进制
    }
    close(fdw);
    close(fdr);
    return 0;

}
```

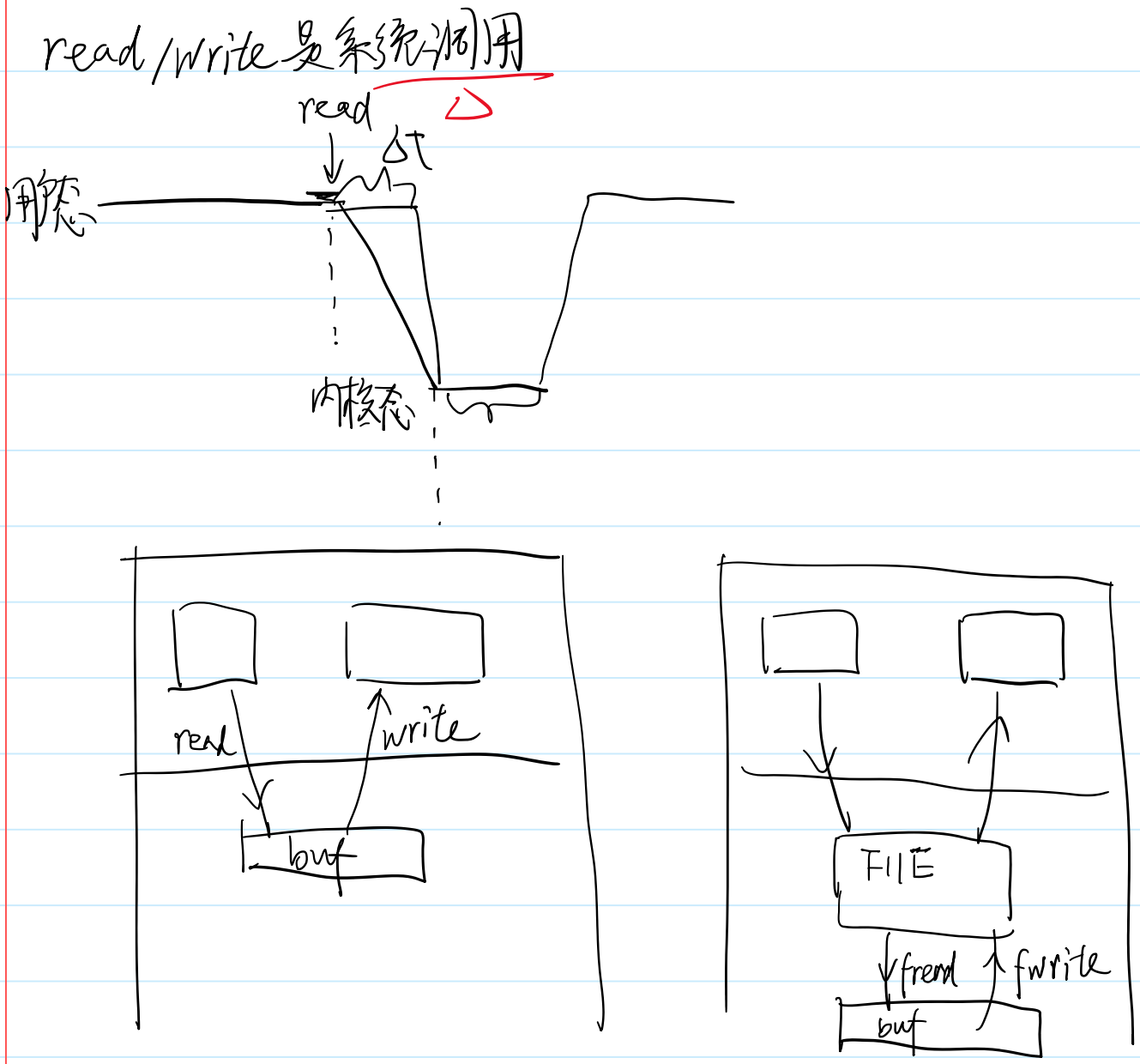
```c
int memcmp(const void *s1, const void *s2, size_t n);
```

2023年8月4日 17:04



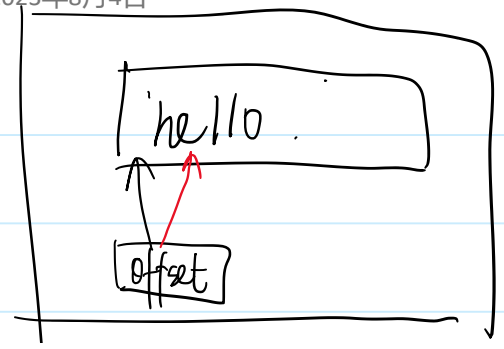
把buf的大小改成1

2023年8月4日 17:17



lseek 更改偏移量

2023年8月4日 17:27



read(fd,buf,1)

```
int main(int argc, char *argv[])
{
    // ./5_toupper file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    char ch;
    while(1){
        ssize_t sret = read(fd,&ch,1);
        if(sret == 0){
            break;
        }
        if(ch >= 'a' && ch <= 'z'){
            ch -= 32;
        }
        lseek(fd,-1,SEEK_CUR);
        write(fd,&ch,1);
    }
    close(fd);
    return 0;
}
```

偏移量 { + 右 / − 左

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

修改的位置

起点

SEEK_SET
TI

SEEK_CUR
TI

SEEK_END
TI

更改文件大小

2023年8月4日 17:42

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

6_ftruncate.c

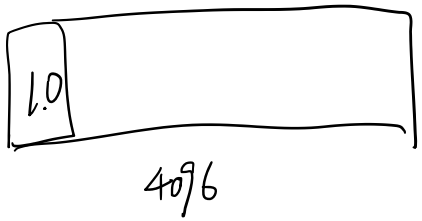
```c
#include <52func.h>
int main(int argc, char *argv[])
{
    // ./6_ftruncate file1
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_RDWR);
    ERROR_CHECK(fd,-1,"open");
    int ret = ftruncate(fd,5);
    ERROR_CHECK(ret,-1,"ftruncate");
    close(fd);
    return 0;
}
```

大→小 保留前面

小→大 填二进制0

文件空洞

2023年8月4日 17:49




```
liao:LinuxDay08$ stat file1
  File: file1
  Size: 10              Blocks: 8 x512     IO Block: 4096   regular file
Device: 805h/2053d      Inode: 2392267     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/    liao)   Gid: ( 1000/    liao)
Access: 2023-08-04 17:47:20.290849491 +0800
Modify: 2023-08-04 17:47:17.898582297 +0800
Change: 2023-08-04 17:47:17.898582297 +0800
 Birth: -
```

```
int ret = ftruncate(fd,40960);
```

```
liao:LinuxDay08$ stat file1
  File: file1
  Size: 40960           Blocks: 8          IO Block: 4096   regular file
Device: 805h/2053d      Inode: 2392267     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/    liao)   Gid: ( 1000/    liao)
Access: 2023-08-04 17:47:20.290849491 +0800
Modify: 2023-08-04 17:52:00.358972501 +0800
Change: 2023-08-04 17:52:00.358972501 +0800
 Birth: -
```