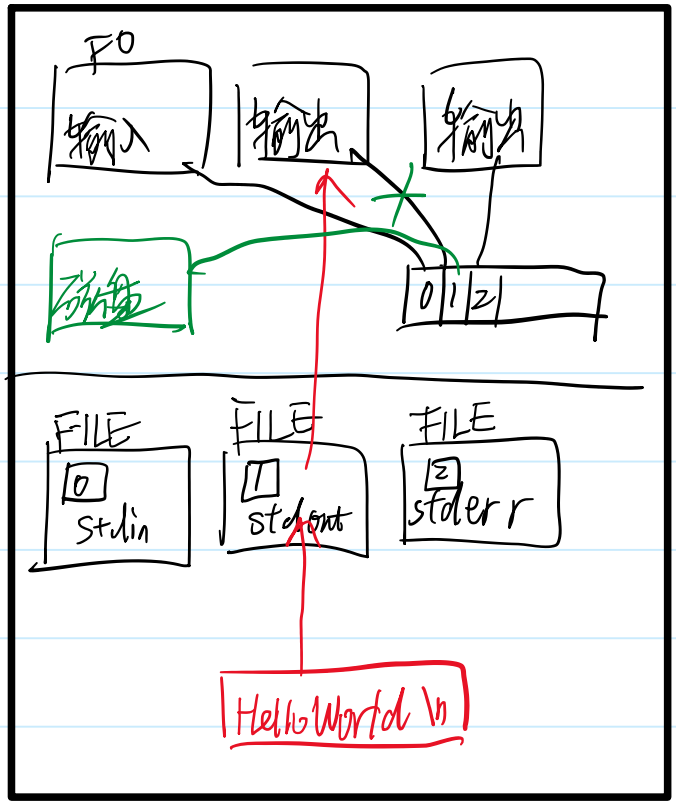
任何程序一启动，会打开3个文件流

2023年8月7日 9:26



printf("Hello World\n");

① 拷贝到stdout

② 缓冲区满/遇到换行，清空stdout

stdout → FO

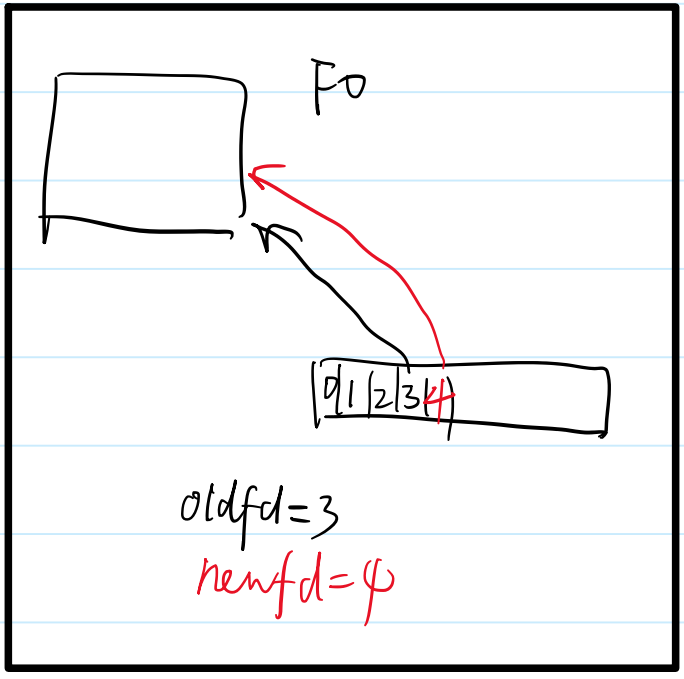
```c
int main(int argc, char *argv[])
{
    // ./1_redirect file1
    printf("Can you see me?\n");
    close(STDOUT_FILENO);
    int fd = open(argv[1],O_WRONLY);
    ERROR_CHECK(fd,-1,"open");
    printf("Can you see me?\n");
    return 0;
}
```

← BUG close()之前记得打印一个换行.

如何先open 再重定向

2023年8月7日 10:04

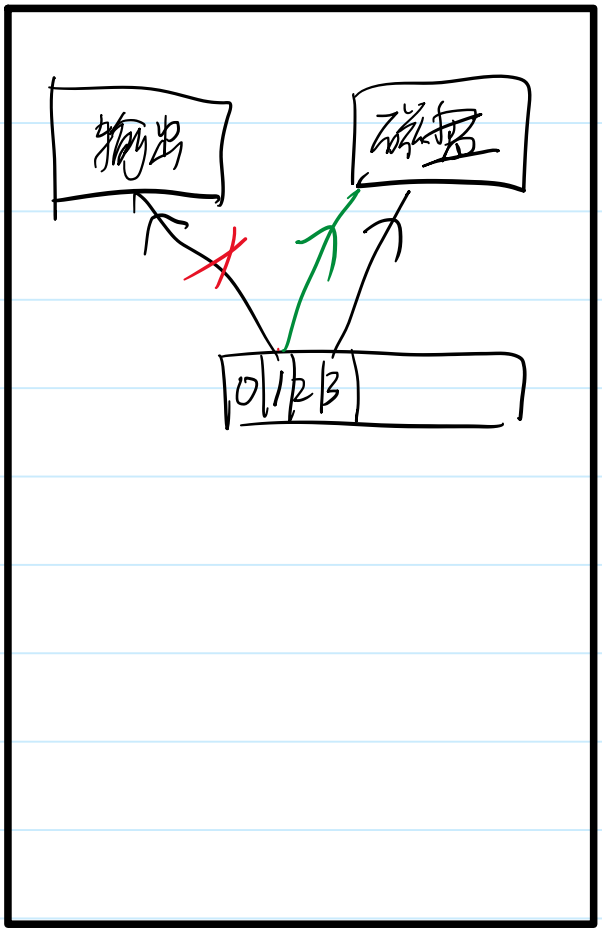
```
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```



dup. 让不同的fd 引用同一个文件对象.

重定向

2023年8月7日 10:09



close(1)

dup(3)

```c
int main(int argc, char *argv[])
{
    // ./2_redirect file2
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_WRONLY);
    ERROR_CHECK(fd,-1,"open");

    printf("You can see me!\n");
    close(STDOUT_FILENO);
    int newfd = dup(fd);
    printf("newfd = %d\n", newfd);
    printf("You cannot see me!\n");
    return 0;
}
```

```
liao:LinuxDay10$ ./2_redirect file2
You can see me!
liao:LinuxDay10$ cat file2
newfd = 1
You cannot see me!
```

引dup之后，close就是关闭fd，fd采用的时数

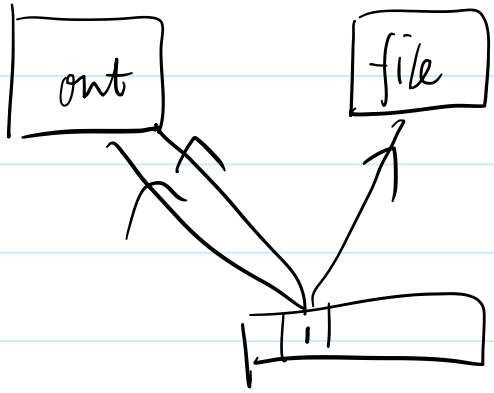
反复横跳

2023年8月7日 10:18

printf 输出

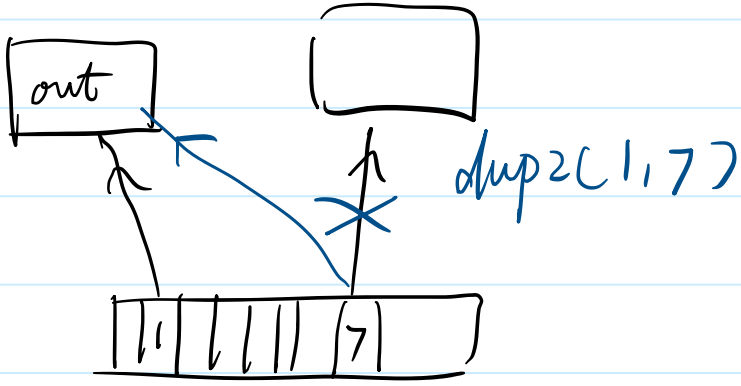
printf 文件

printf 输出

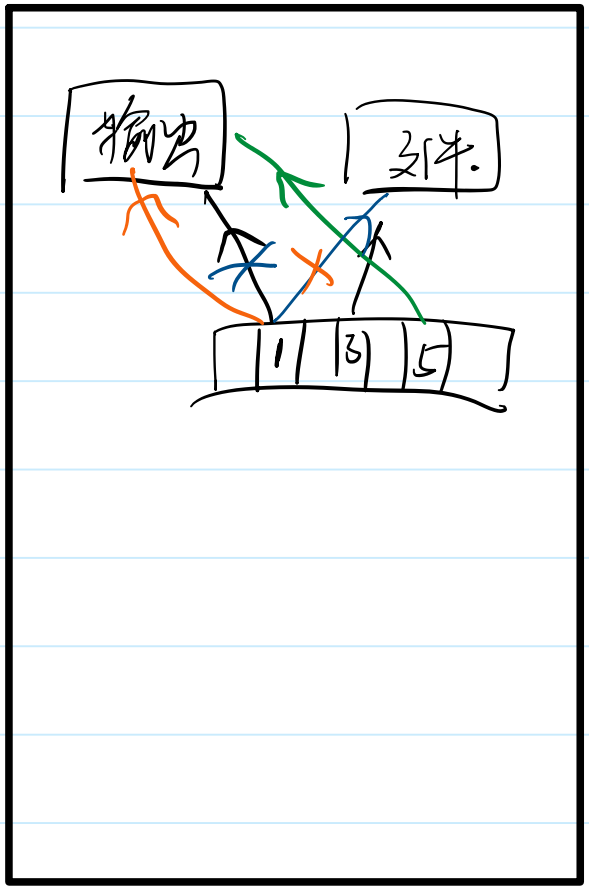



```
int dup2(int oldfd, int newfd);
```

重定向



2023年8月7日 10:25



① printf 输出

备份输出 dup2(1,5)

让1去引用文件 dup2(3,1)

② printf 文件

让1去引用输出 dup2(5,1)

③ printf. 输出

2023年8月7日 10:33

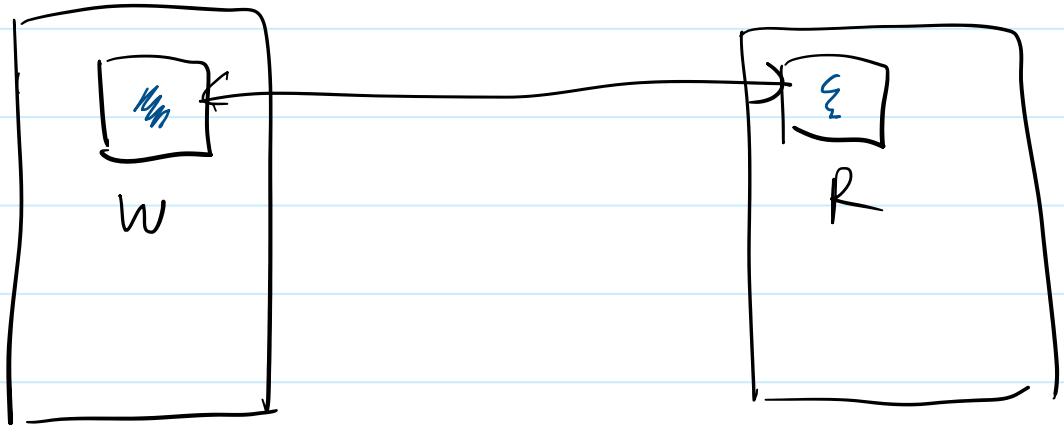
```c
int main(int argc, char *argv[])
{
    // ./3_redirect file3
    ARGS_CHECK(argc,2);
    int fd = open(argv[1],O_WRONLY);
    ERROR_CHECK(fd,-1,"open");

    printf("我过来啦\n");
    int backup_fd = 10;
    dup2(STDOUT_FILENO,backup_fd);
    dup2(fd,STDOUT_FILENO);
    printf("我回去了\n");
    dup2(backup_fd,STDOUT_FILENO);
    printf("我又过来啦！ \n");
    return 0;
}
```

有名管道

2023年8月7日 10:57

named pipe / FIFO 进程间通信机制



MKFIFO(1)

NAME

mkfifo - make FIFOs (named pipes)

```
liao:LinuxDay10$ mkfifo 1.pipe
liao:LinuxDay10$ ll
total 128
drwxrwxr-x  2 liao liao  4096 Aug  7 11:02 ./
drwxrwxr-x 11 liao liao  4096 Aug  7 09:43 ../
-rwxrwxr-x  1 liao liao 20240 Aug  7 09:50 0_mmap*
-rw-rw-r--  1 liao liao   562 Aug  7 09:50 0_mmap.c
prw-rw-r--  1 liao liao     0 Aug  7 11:02 1.pipe
```

管道的特点

2023年8月7日 11:03

```
liao:LinuxDay10$ cat 1.pipe
hello
```

echo hello > 1.pipe

管道只用于通信，不能存储数据；不能用 vim 打开.

单工. A ⟶ B

✓ 半双工 A ⟶ B or A ⟵ B 不能同时.

全双工 A ⟷ B. 任何时刻

虽然是半双工，但是用户一般当单工使用.

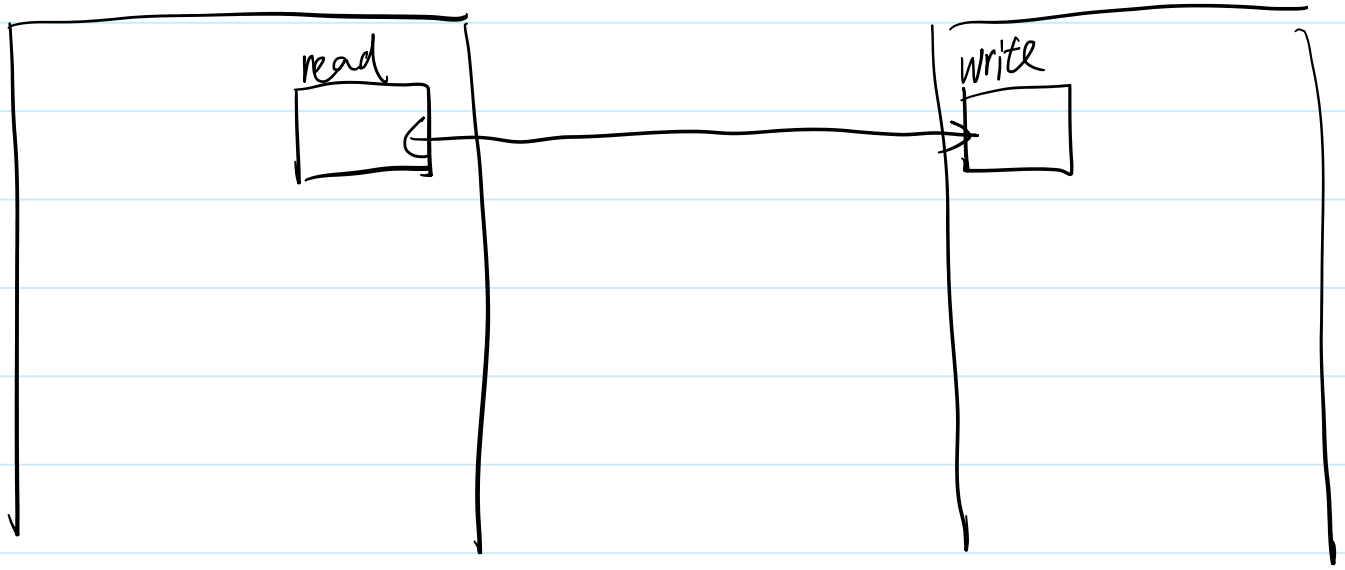
非要用双工，使用两根管道.

open

2023年8月7日 11:13

open("1.pipe", O_RDONLY);

O_WRONLY



open会引发阻塞

2023年8月7日 11:18

```c
int main(int argc, char *argv[])
{
    // ./4_open_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;
}
```

```c
int main(int argc, char *argv[])
{
    // ./4_open_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;
}
```

open管道的一端会阻塞，阻塞直到另一端被打开.

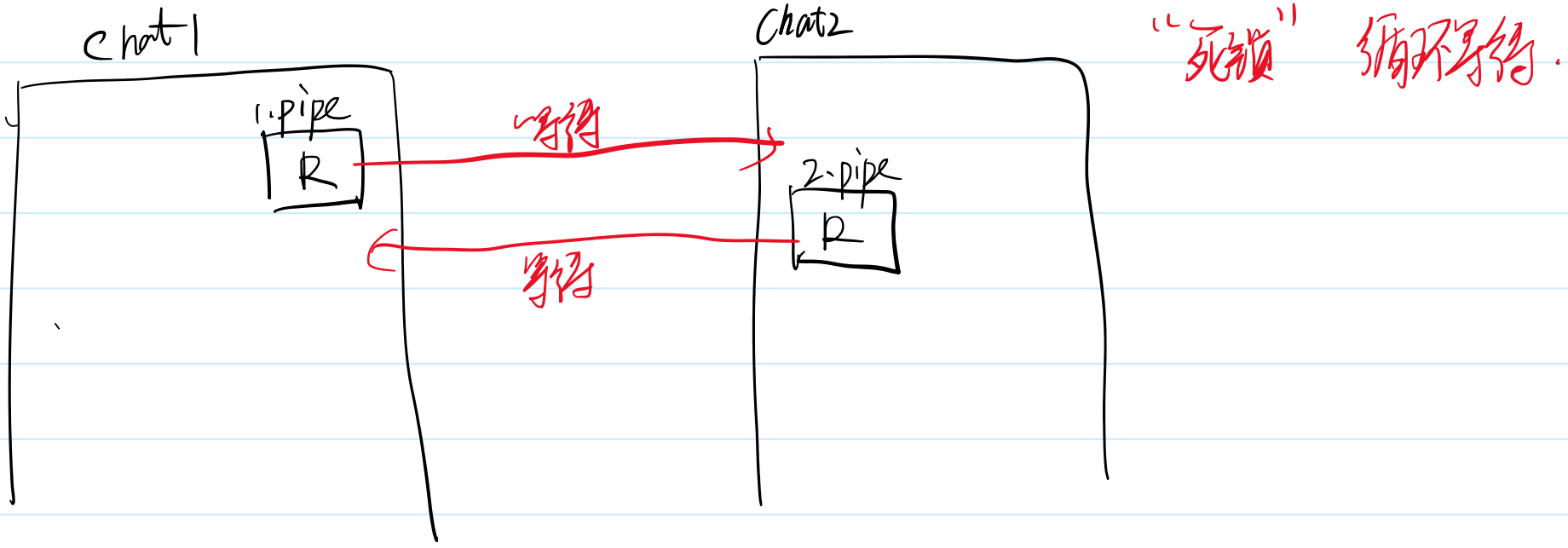
死锁

2023年8月7日 11:27

解决方案：调整获取资源的顺序.

```c
int main(int argc, char *argv[])
{
    // ./5_open_chat1 1.pipe 2.pipe
    ARGS_CHECK(argc,3);
    int fdr = open(argv[1],O_RDONLY);
    int fdw = open(argv[2],O_WRONLY);
    printf("chat1 is OK!\n");
    return 0;
}
```

```c
int main(int argc, char *argv[])
{
    // ./5_open_chat2 2.pipe 1.pipe
    ARGS_CHECK(argc,3);
    int fdr = open(argv[1],O_RDONLY);
    int fdw = open(argv[2],O_WRONLY);
    printf("chat2 is OK!\n");
    return 0;
}
```



"死锁" 编不写得.

死锁的解决

2023年8月7日 11:35

```c
int main(int argc, char *argv[])
{
    // ./5_open_chat2 1.pipe 2.pipe
    ARGS_CHECK(argc,3);
    int fdw = open(argv[1],O_WRONLY);
    int fdr = open(argv[2],O_RDONLY);
    printf("chat2 is OK!\n");
    return 0;
}
```

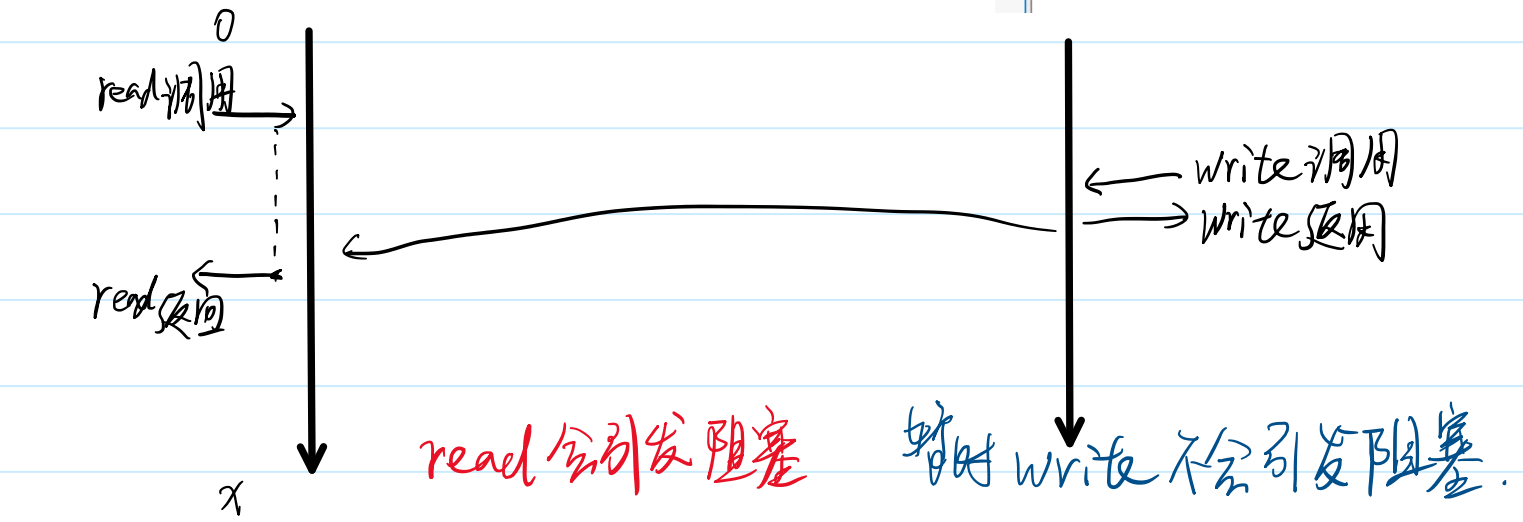
read write

2023年8月7日 11:36

管道文件的read/write 类似于设备文件.

```c
int main(int argc, char *argv[])
{
    // ./6_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");
    char buf[4096] = {0};
    ssize_t sret = read(fdr,buf,sizeof(buf));
    printf("sret = %ld, buf = %s\n", sret, buf);
    close(fdr);
    return 0;
}
```

```c
int main(int argc, char *argv[])
{
    // ./6_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");
    sleep(5);
    printf("sleep over!\n");
    write(fdw,"howareyou",9);
    close(fdw);
    return 0;
}
```

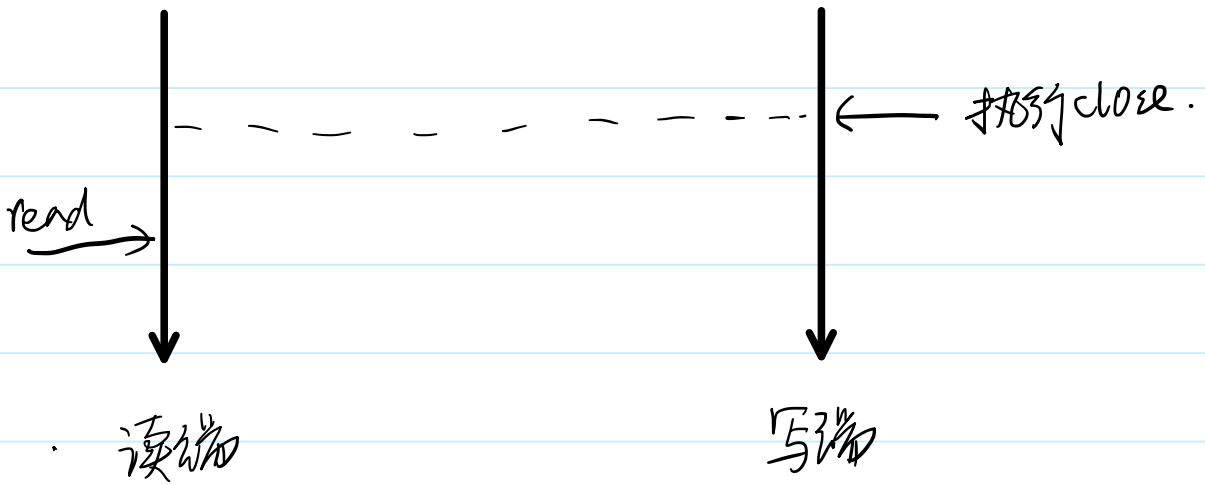


read会引发阻塞

暂时write不会引发阻塞.

close的特点

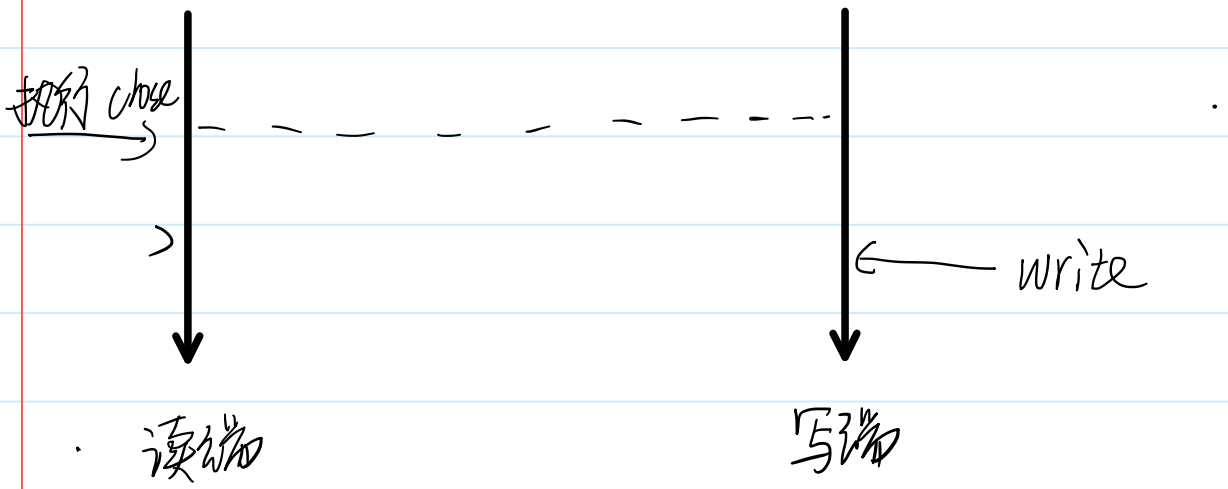
2023年8月7日 11:50



现象：缓冲区无数据，写端已经close，读端read，不会阻塞返回值为0。

读端先关闭 写端继续写

2023年8月7日 14:33



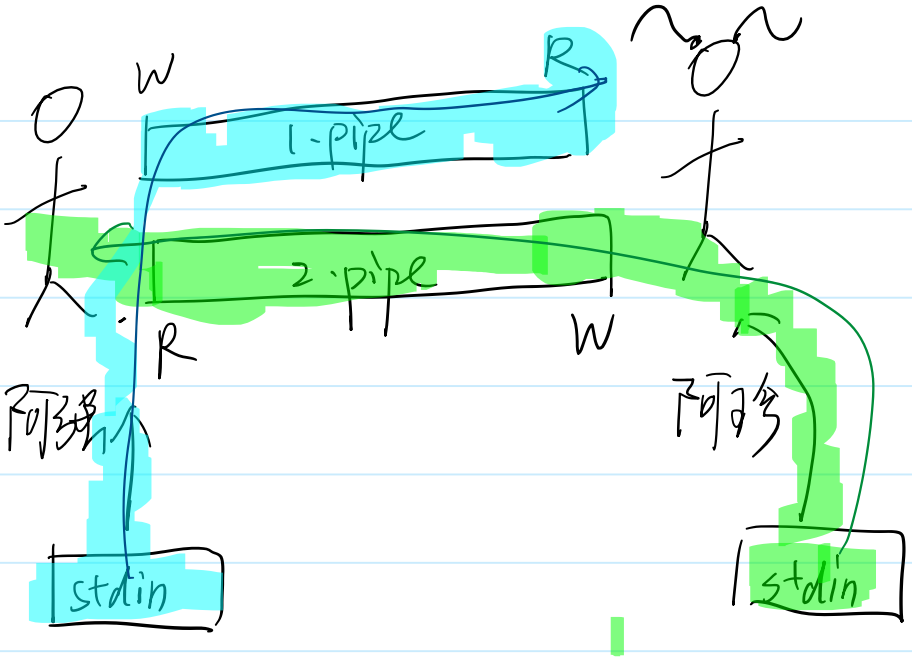

```
Program received signal SIGPIPE, Broken pipe.
0x00007ffff7ed2077 in __GI___libc_write (fd=3, buf=0x555555556038,
    nbytes=9) at ../sysdeps/unix/sysv/linux/write.c:26
26      ../sysdeps/unix/sysv/linux/write.c: No such file or directory.
(gdb)
```

读端已经close，写端继续执行write → 触发SIGPIPE信号

异常终止

即时聊天

2023年8月7日 14:46



2023年8月7日 15:02

8_azhen.c buffers

```c
#include <52func.h>
int main(int argc, char *argv[])
{
    // ./8_azhen 1.pipe 2.pipe
    ARGS_CHECK(argc,3);
    int fdr = open(argv[1],O_RDONLY);
    int fdw = open(argv[2],O_WRONLY);
    printf("azhen is ready!\n");
    char buf[4096];
    while(1){
        memset(buf,0,4096);
        read(fdr,buf,4096);
        printf("buf = %s\n", buf);

        memset(buf,0,4096);
        read(STDIN_FILENO,buf,4096);
        write(fdw,buf,strlen(buf));
    }
    return 0;
}
```

读管道

读stdin

8_aqiang.c

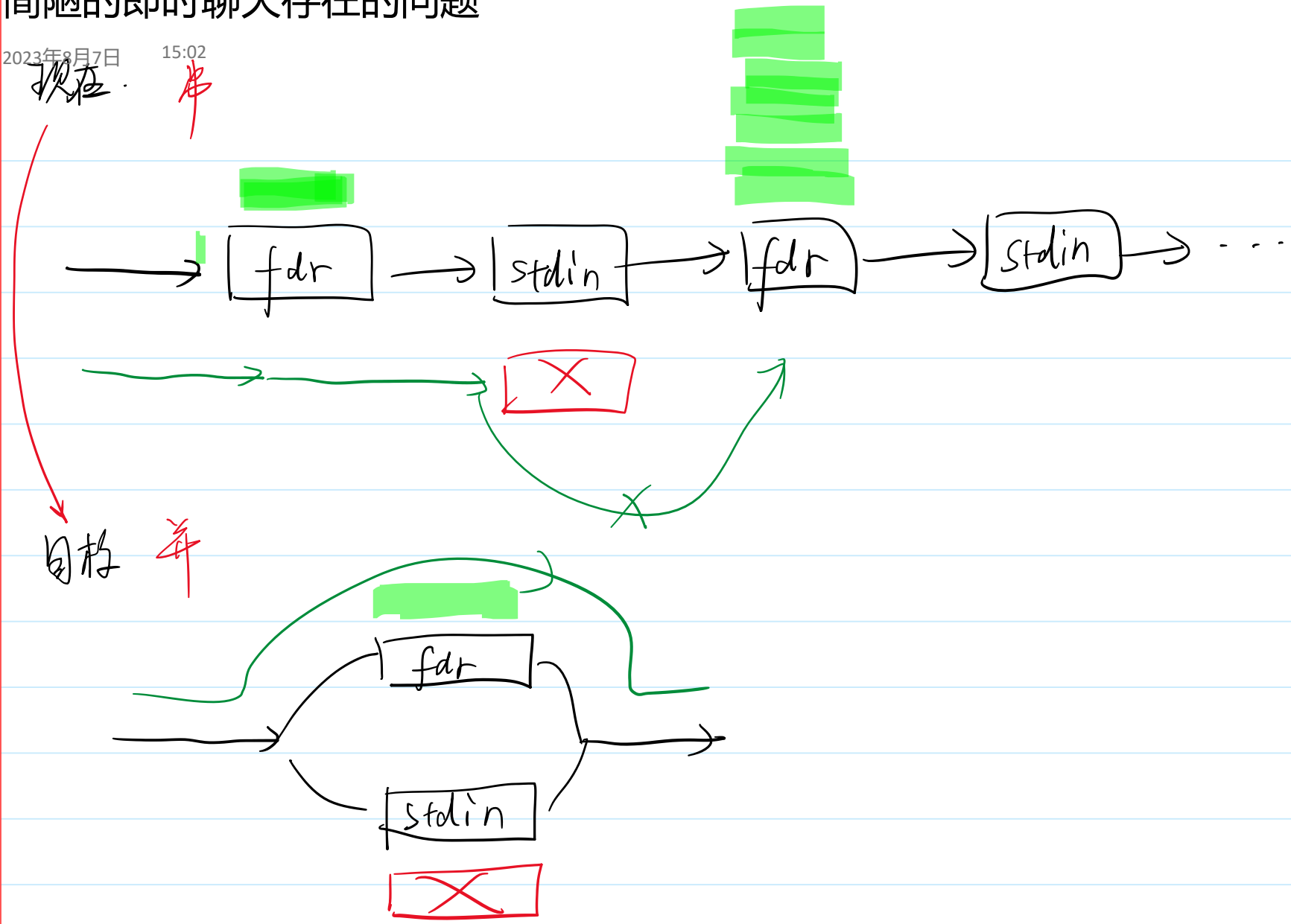
```c
#include <52func.h>
int main(int argc, char *argv[])
{
    // ./8_aqiang 1.pipe 2.pipe
    ARGS_CHECK(argc,3);
    int fdw = open(argv[1],O_WRONLY);
    int fdr = open(argv[2],O_RDONLY);
    printf("aqiang is ready!\n");
    char buf[4096];
    while(1){
        memset(buf,0,4096);
        read(STDIN_FILENO,buf,4096);
        write(fdw,buf,strlen(buf));

        memset(buf,0,4096);
        read(fdr,buf,4096);
        printf("buf = %s\n", buf);
    }

    return 0;
}
```

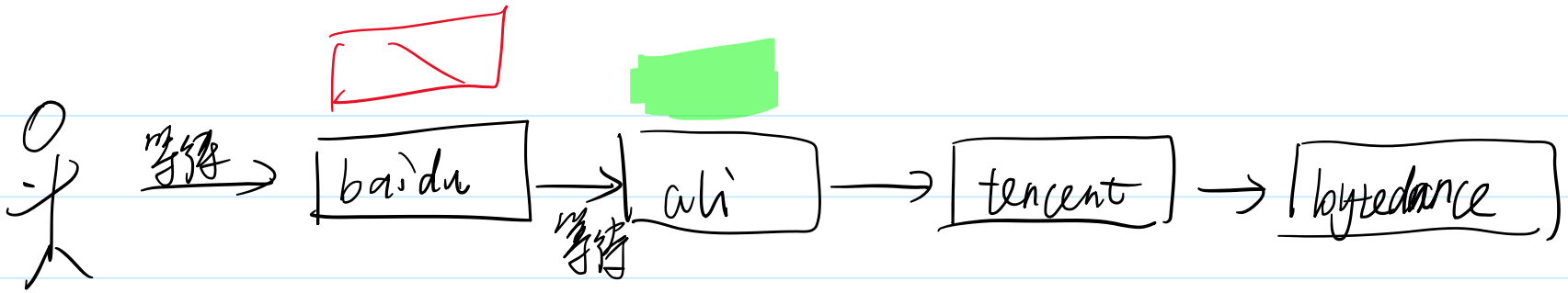
简陋的即时聊天存在的问题

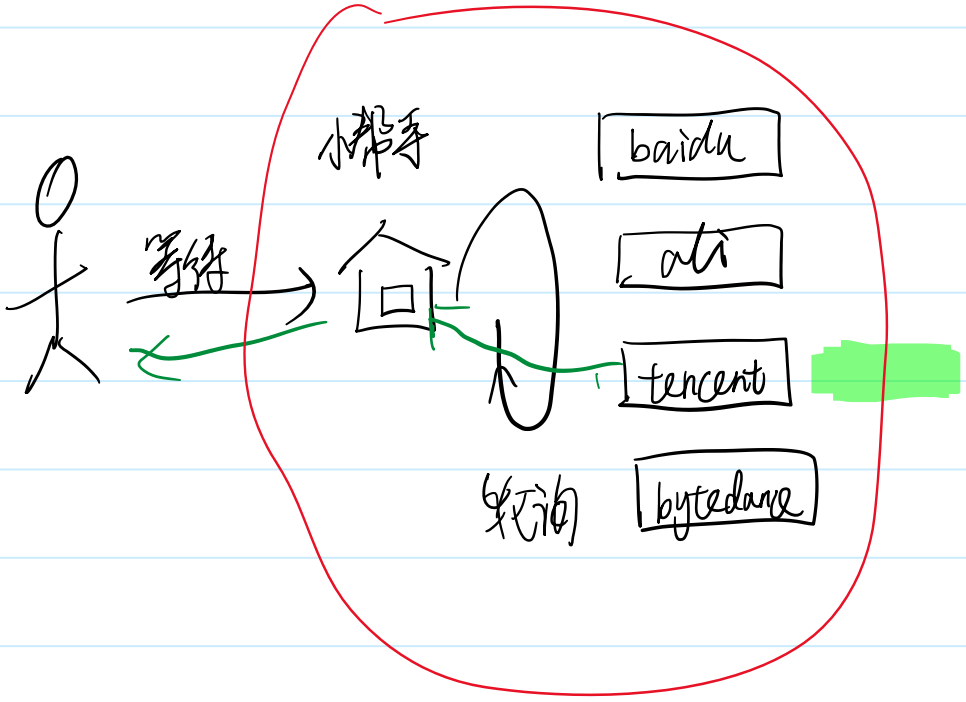
2023年8月7日 15:02



IO多路复用

2023年8月7日 15:09





select

2023年8月7日 15:25

synchronous I/O multiplexing

监听集合："小弟"需要观察的资源

1、在内核中创建监听集合 fd_set

```
void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

FD_CLR → 去掉一个
FD_ISSET → 检查是否在里面
FD_SET → 增加一个
FD_ZERO → 初始化

2. FD_ZERO 初始化集合
3. FD_SET 增加监听

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

4. 进程调用 select a. OS 在做轮询
   b. 进程在等待
   b. 一旦某个fd就绪，select返回，携带就绪集合 FD_ISSET

select函数的设计

2023年8月7日 16:05

传入传出参数：select 调用过程会修改内容.

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

监听集合 select返回 子集集合

nfds ⟹ 所有监听集合里面，最大数值的fd再加1.

readfds / writefds / exceptfds

没有就填NULL

→ 读事件.监听集合的地址

timeout 永久等待 填NULL

代码

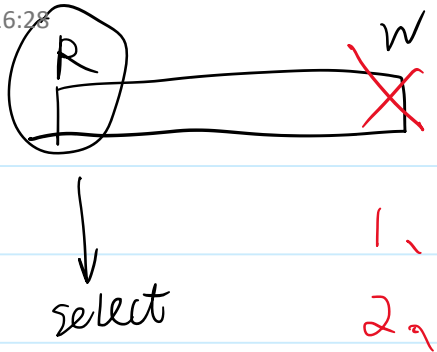
2023年8月7日 16:19

```
char buf[4096];
// 1. 为监听集合申请内存
fd_set rdset;
while(1){
    // 2. 初始化监听集合
    FD_ZERO(&rdset);
    // 3. 增加监听 fdr stdin
    FD_SET(STDIN_FILENO,&rdset);
    FD_SET(fdr,&rdset);
    // 4. OS轮询资源是否就绪，进程陷入阻塞
    select(fdr+1,&rdset,NULL,NULL,NULL);
    // nfds 最大的文件描述符+1
    // rdset select调用前是监听集合，select返回后是就绪集合
    // 如果需要多次调用select，每次调用之前要重置一下监听集合
    if(FD_ISSET(STDIN_FILENO,&rdset)){
        memset(buf,0,sizeof(buf));
        read(STDIN_FILENO,buf,sizeof(buf));
        write(fdw,buf,strlen(buf));
    }
    if(FD_ISSET(fdr,&rdset)){
        memset(buf,0,sizeof(buf));
        read(fdr,buf,sizeof(buf));
        printf("buf = %s\n", buf);
    }
}
```

聊天关闭的时候

2023年8月7日 16:28

```
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
buf =
```



1、W端关闭了.

2、如果R做read. 立刻返回

↓

称为一种状态.

3、select监听到R，R状态则select状态

↓

改变了.

兼容聊天关闭

2023年8月7日 16:40

① ctrl+d. 主动退出

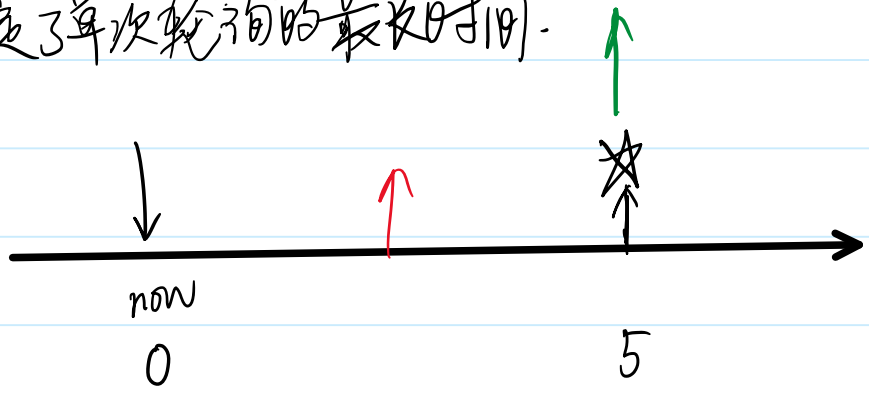
② read fdr 返回0. 对方退出

```
if(FD_ISSET(STDIN_FILENO,&rdset)){
    memset(buf,0,sizeof(buf));
    ssize_t sret = read(STDIN_FILENO,buf,sizeof(buf));
    if(sret == 0){
        //主动退出
        write(fdw,"nishigehaoren",13);
        break;
    }
    write(fdw,buf,strlen(buf));
}
if(FD_ISSET(fdr,&rdset)){
    memset(buf,0,sizeof(buf));
    ssize_t sret = read(fdr,buf,sizeof(buf));
    if(sret == 0){
        //对方退出
        printf("Hehe\n");
        break;
    }
    printf("buf = %s\n", buf);
}
```

select的超时

2023年8月7日 17:03

指定了单次轮询的最长时间.



timeout 3s.

timeout 0s

传值传参数.

select返回时，timeout返回剩余时间.

```
struct timeval *timeout
```

```
struct timeval {
    long    tv_sec;         /* seconds */
    long    tv_usec;        /* microseconds */
};
```

timeout = tv_sec 秒 + tv_usec 微秒

time函数

2023年8月7日 17:11

时间戳/系统时间 1970.1.1 每过1s 加1.

精度为秒.

日历时间.

```
char *ctime(const time_t *timep);
```

固定格式日历时间.

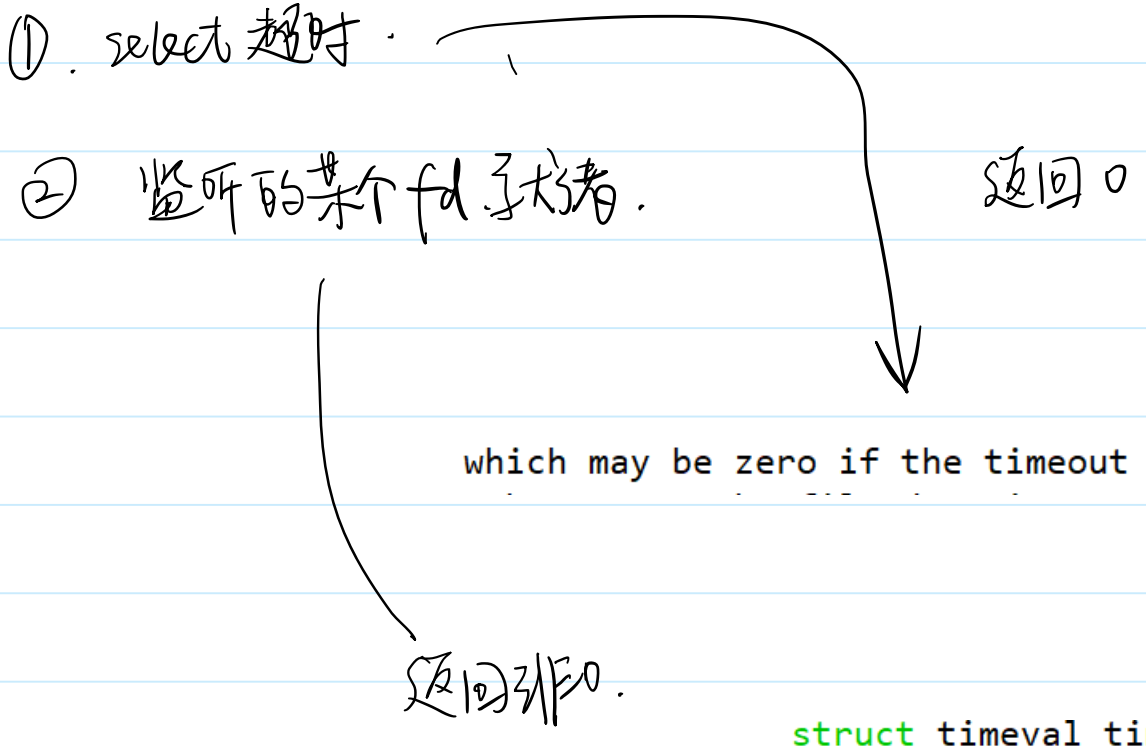
```
struct tm *localtime(const time_t *timep);
```

日历时间各个分量

```
struct tm {
    int tm_sec;    /* Seconds (0-60) */
    int tm_min;    /* Minutes (0-59) */
    int tm_hour;   /* Hours (0-23) */
    int tm_mday;   /* Day of the month (1-31) */
    int tm_mon;    /* Month (0-11) */
    int tm_year;   /* Year - 1900 */
    int tm_wday;   /* Day of the week (0-6, Sunday = 0) */
    int tm_yday;   /* Day in the year (0-365, 1 Jan = 0) */
    int tm_isdst;  /* Daylight saving time */
};
```

区分返回的原因

2023年8月7日 17:21



which may be zero if the timeout expires before anything interest-

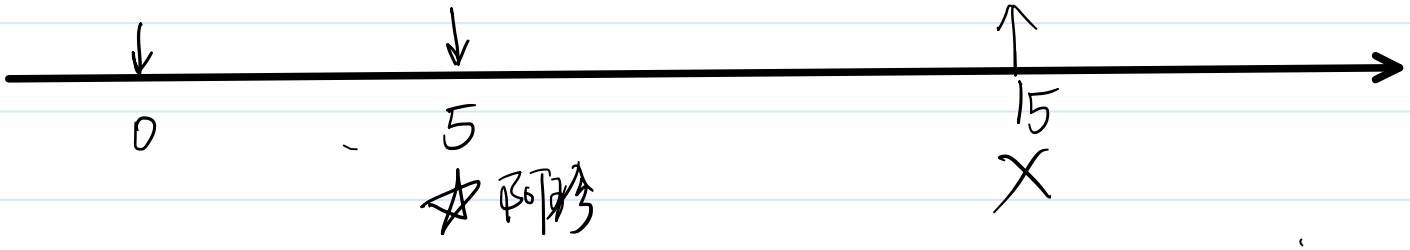
```c
struct timeval timeout;
timeout.tv_sec = 2;
timeout.tv_usec = 0;
int ret = select(fdr+1,&rdset,NULL,NULL,&timeout);
if(ret == 0){
    time_t now = time(NULL);
    printf("timeout! curtime = %s\n", ctime(&now));
    continue;
}
```

超时踢人

2023年8月7日 17:27

阿妙和阿强聊天. 阿强10s未发言, 阿妙踢掉阿强.

可能的方案一. timeout 固定10s. ✗



方案二. timeout不重置. 最初或发言才重置. ✓ 只适用于一个阿强

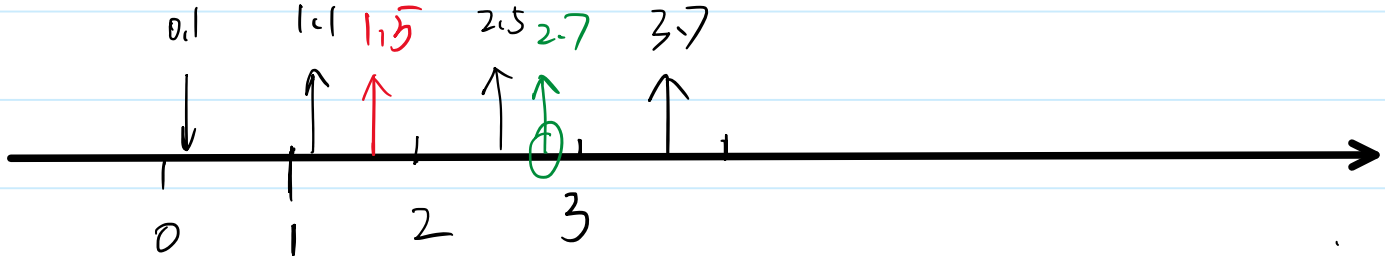
超时踢人解决方案3

2023年8月7日 17:32

物性精度 10~11s 之间浮动

①timeout 设置为1秒.

② 每次就诊记录当前时间 time(NULL)



→ 超时

→ 预警

→ 报警

0~1、1~2、2~3、3~4、.... 秒就诊一次.

写入也会阻塞

2023年8月7日 17:50

10_read_write_block.c buffers

```c
#include <52func.h>
int main(int argc, char *argv[])
{
    // ./10_read_write_block 1.pipe
    ARGS_CHECK(argc,2);
    int fdr = open(argv[1],O_RDONLY);
    sleep(10);
    printf("sleep over!\n");
    char buf[4096];
    read(fdr,buf,sizeof(buf));
    sleep(100);
    close(fdr);
    return 0;
}
```

10_write_write_block.c

```c
#include <52func.h>
int main(int argc, char *argv[])
{
    // ./10_write_write_block 1.pipe
    ARGS_CHECK(argc,2);
    int fdw = open(argv[1],O_WRONLY);
    char buf[4096] = {0};
    int cnt = 0;
    ssize_t sret;
    ssize_t total = 0;
    while(1){
        sret = write(fdw,buf,sizeof(buf));
        total += sret;
        printf("cnt = %d, total = %ld\n", cnt++, total);
    }
    return 0;
}
```