学习目标

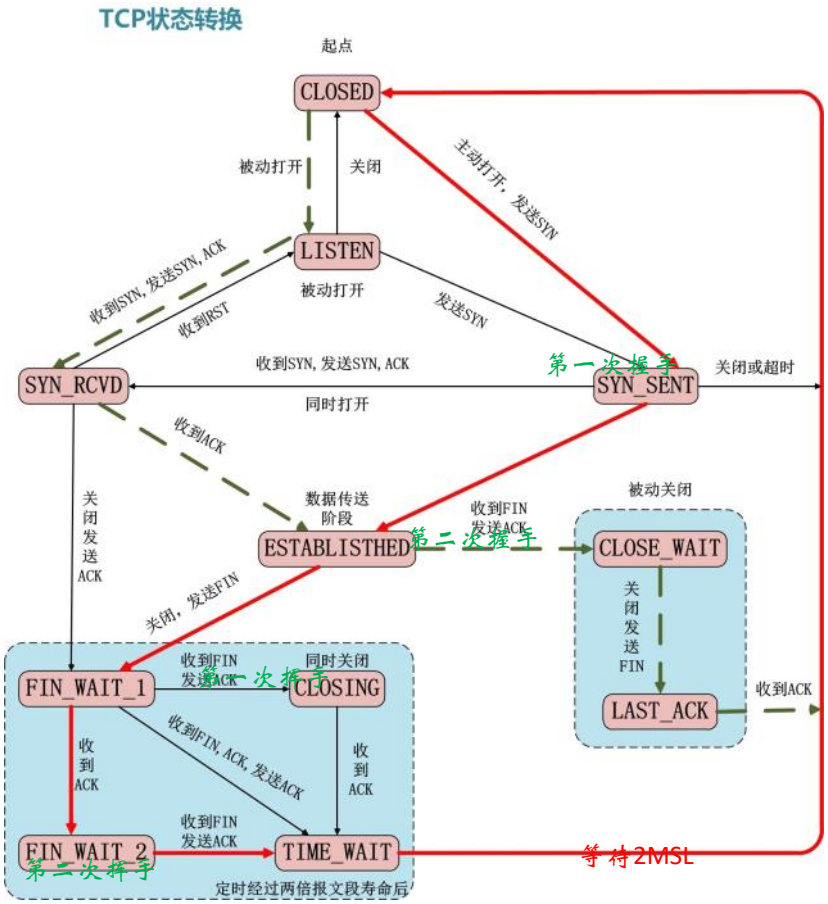
1. 能够描述TCP通信过程中主要状态
2. 独立使用select实现IO多路转接
3. 理解使用poll实现IO多路转接操作流程

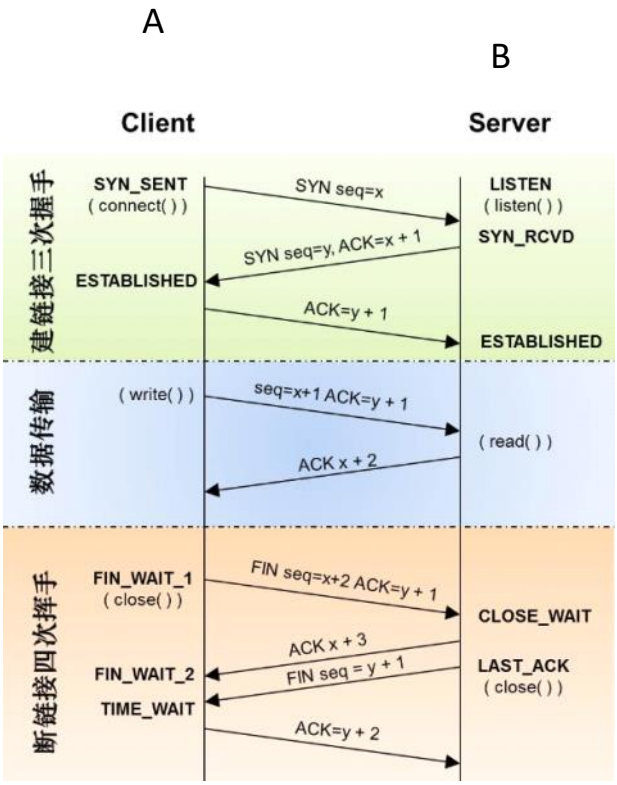
0 - send/recv

TCP通信

1. 数据接收
   - ssize_t read(int fd,   void *buf, size_t count);
   - ssize_t recv(int sockfd, void *buf, size_t len, int flags);
     - flags = 0
2. 数据发送
   - ssize_t write(int fd,   const void *buf, size_t count);
   - ssize_t send(int sockfd, const void *buf, size_t len, int flags);

1 - tcp状态转换



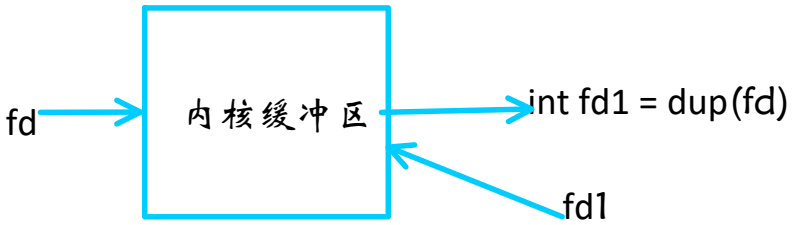


第3、4次挥手

1. 半关闭
   - 如何理解?
     - A给B主动发送FIN, 但是B没有给A发送FIN
     - A关闭了与B的连接, B还处于与A的连接状态
   - 特点:
     - A可以接收B发送过来的数据
     - A不能给B发送数据
   - 函数: int shutdown(int sockfd, int how);
     - sockfd: 要半关闭的一方对应的文件描述符
       - 通信的文件描述符
     - how:
       - SHUT_RD - 0 - 读
       - SHUT_WR - 1 - 写
       - SHUT_RDWR - 2 - 读写
   - 思考: close函数能否实现半关闭?
     - 只考虑关闭写操作

- 是不是永远都能成功？
  - 如果对文件描述符做了dup或dup2操作

2. 2MSL
   a. 等待时长
   b. 主动关闭连接的一方，处于TIME_WAIT状态
   c. 有的地方：2分钟，30s，一般时候是30s(MSL)
3. 查看网络相关状态信息
   - 命令：netstat
   - 参数：
     -a (all)显示所有选项，默认不显示LISTEN相关
     -p 显示建立相关链接的程序名
     -n 拒绝显示别名，能显示数字的全部转化成数字。
     -t (tcp)仅显示tcp相关选项
     -u (udp)仅显示udp相关选项
     -l 仅列出有在 Listen (监听) 的服务状态



close(fd); - 只能关闭fd
shutdown(fd)
同时关闭fd和fd1

2 - 端口复用

端口复用最常用的用途是:

- 防止服务器重启时之前绑定的端口还未释放
- 程序突然退出而系统没有释放端口

设置方法:

```
int opt = 1;
setsockopt(sockfd, SOL_SOCKET,
                SO_REUSEADDR,
                (const void *)&opt, sizeof(opt));
```

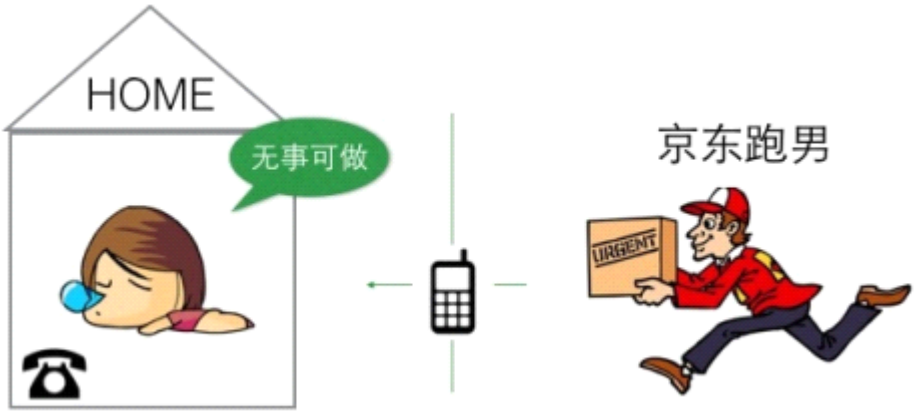
注意事项:

- 要在绑定之前做该设置

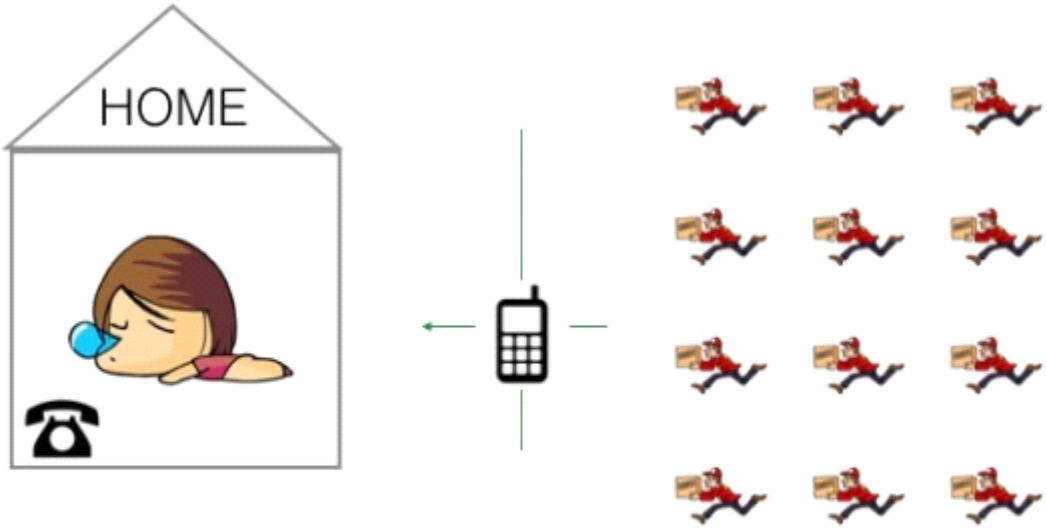
3 - IO多路转接

1. IO操作方式

- 阻塞等待
  - 好处: 不占用cpu宝贵的时间片



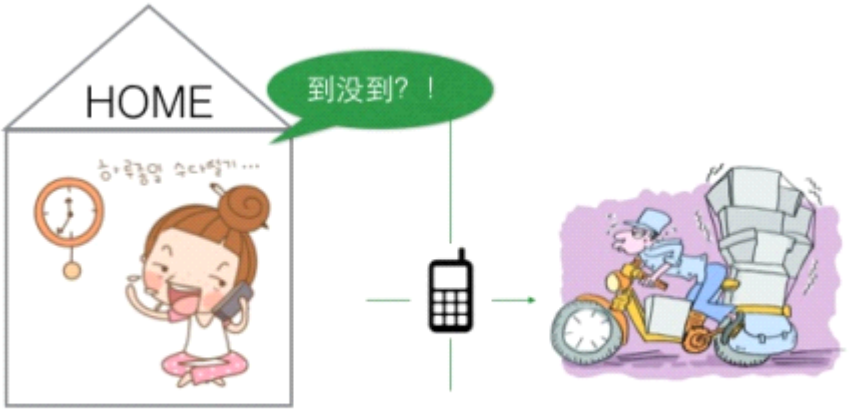
  - 缺点: 同一时刻只能处理一个操作, 效率低



如果同一时刻到达，你同一时刻可能只签收并验货一份快递
你的电话是座机，在你签收的时候，便接不到其他快递员的电话。

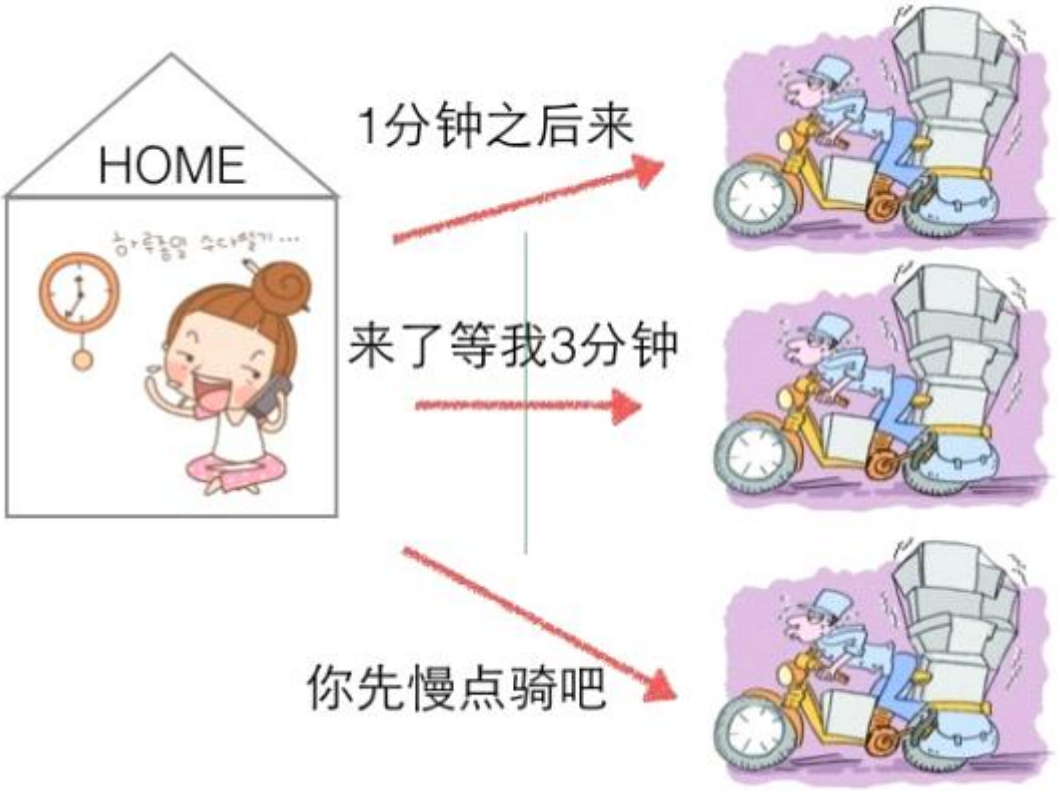
多线程 或 多进程

- 非阻塞, 忙轮询
  - 优点: 提高了程序的执行效率
  - 缺点: 需要占用更多的cpu和系统资源
    - 一个任务

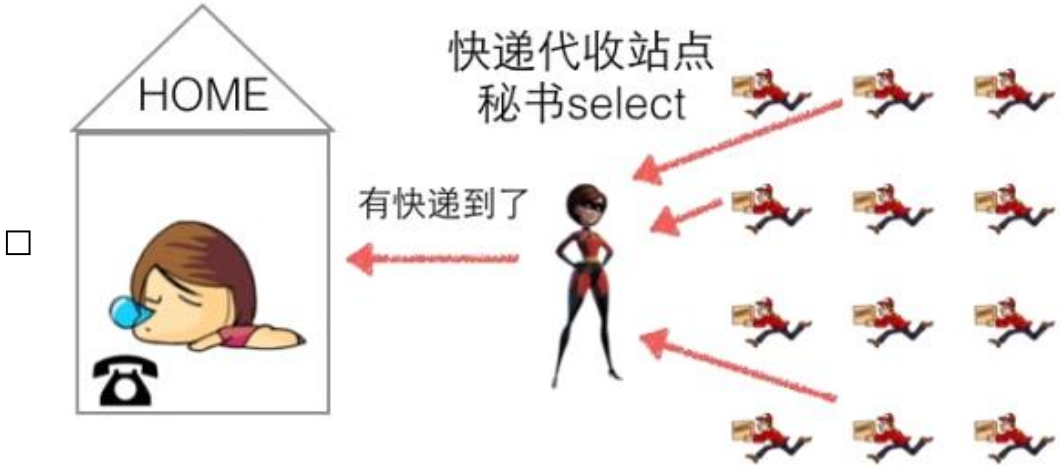


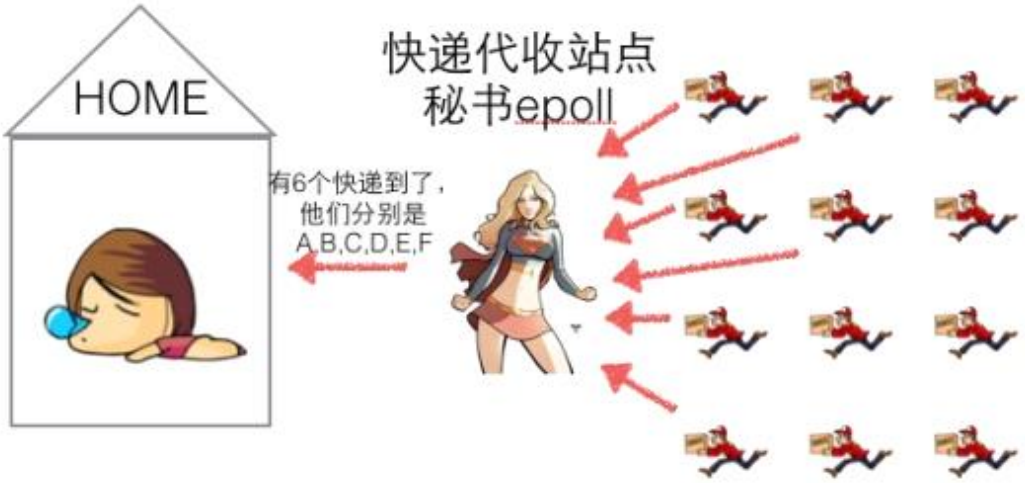
每隔一分钟催一次

- 多个任务



- 解决方案：使用IO多路转接技术 select/poll/epoll
  - 第一种：select/poll



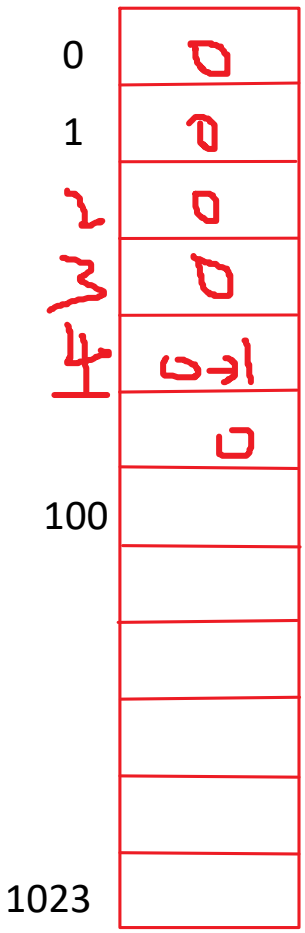
- select 代收员比较懒，她只会告诉你有几个快递到了，但是哪个快递，你需要挨个快递员问一遍。
  - 第二种：epoll
    - 

    - epoll代收快递员很勤快，她不仅会告诉你有几个快递到了，还会告诉你是哪个快递公司的快递.

2. 什么是I/O多路转接技术：
   - 先构造一张有关文件描述符的列表，将要监听的文件描述符添加到该表中
   - 然后调用一个函数，监听该表中的文件描述符，直到这些描述符表中的一个进行I/O操作时，该函数才返回。
     - 该函数为阻塞函数
     - 函数对文件描述符的检测操作是由内核完成的
   - 在返回时，它告诉进程有多少(哪些)描述符要进行I/O操作。

4 - IO 多路转接 - select

fd_set;




```
struct timeval {
    long    tv_sec;
    long    tv_usec;
};
```

sigset_t myset;

函数原型：

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

- 参数：
  - nfds: 最大读文件描述符值+1
  - readfds: 要检测的读集合
    - 接收数据
    - 数据中有1024个标志位
    - 传入传出参数
    - 内核会对这个表做修改
  - writefds: NULL
  - exceptfds: NULL
  - timeout: 设置阻塞的时间
    - 设置不阻塞：
      - timeval val;
      - val.tv_sec = 0;
      - val.tv_usec= 0;
    - 让函数一直阻塞，当检测的文件描述符对应的缓冲区发送变化的时候，解除阻塞 NULL
- 返回值：
  - 告诉用户有几个文件描述符缓冲区发生了变化

文件描述符集类型：fd_set rdset;

文件描述符操作函数：

- 全部清空
  - void FD_ZERO(fd_set *set);
- 从集合中删除某一项

    - void FD_CLR(int fd, fd_set *set);
- 将某个文件描述符添加到集合
    - void FD_SET(int fd, fd_set *set);
- 判断某个文件描述符是否在集合中
    - int FD_ISSET(int fd, fd_set *set);

使用select函的优缺点:

- 优点:
    - 跨平台
- 缺点:
    - 每次调用select，都需要把fd集合从用户态拷贝到内核态，这个开销在fd很多时会很大
    - 同时每次调用select都需要在内核遍历传递进来的所有fd，这个开销在fd很多时也很大
    - select支持的文件描述符数量太小了，默认是1024

select实现IO转接思路

1. select函数判断文件描述符是否有动作
2. 当有读动作，判断
3. 初始化读集合的时候：
   a. fd_set readfds, test1;
   b. FD_ZERO(&readfds);
   c. FD_SET(lfd, &readfds);
   d. int maxfd = lfd;

每调用一次select，如果集合中 文件描述符没有IO变化
这个文件文件描述符对用标志位会被清空

```
while(1)
{
    test1 = reads;
    select(maxfd+1, &test1, NULL, NULL, NULL);
    // 判断
    if(监听到了连接请求)
    {
        int cfd = accept();
        // cfd放到监听读的集合中
        FD_SET(cfd, reads);
        // 更新最大文件描述符
        maxfd = maxfd < cfd ? cfd : maxfd;
    }
    // 客户端给我发数据
    for(int i = lfd+1; i<=maxfd; ++i)
```

```
    {
        // 遍历
        int fd = i;
        // 判断哪一个有读操作
        if(FD_ISSET(fd, &reads);
        {
            // 读数据
            read/recv
            // 发数据
            send/write
        }
    }
}
close(lfd);
```

5 - IO多路转接 - poll

poll结构体:

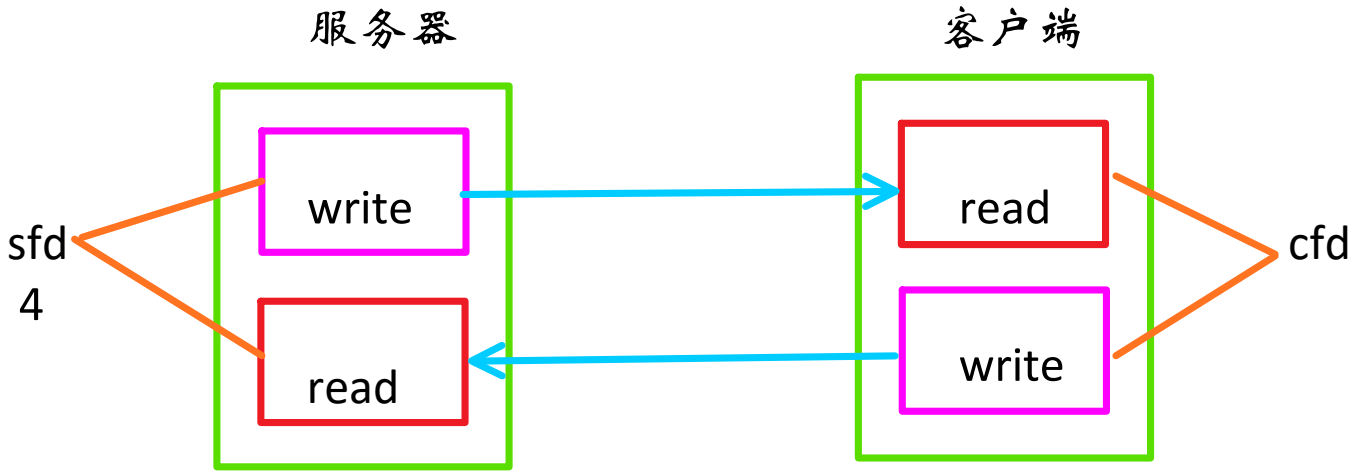
```
struct pollfd {
    int   fd;         /* 文件描述符 */
    short events;     /* 等待的事件 */
    short revents;    /* 实际发生的事件 */
};
```

事件	常值	作为events的值	作为revents的值	说明
读事件	POLLIN	✔	✔	普通或优先带数据可读
	POLLRDNORM	✔	✔	普通数据可读
	POLLRDBAND	✔	✔	优先级带数据可读
	POLLPRI	✔	✔	高优先级数据可读
写事件	POLLOUT	✔	✔	普通或优先带数据可写
	POLLWRNORM	✔	✔	普通数据可写
	POLLWRBAND	✔	✔	优先级带数据可写
错误事件	POLLERR		✔	发生错误
	POLLHUP		✔	发生挂起
	POLLNVAL		✔	描述不是打开的文件

函数原型:

- int select(int nfds,
  fd_set *readfds,
  fd_set *writefds,
  fd_set *exceptfds,
  struct timeval *timeout);
- int poll(struct pollfd *fd, nfds_t nfds, int timeout);
  - pollfd -- 数组的地址
  - nfds: 数组的最大长度, 数组中最后一个使用的元素下标+1
    - 内核会轮询检测fd数组的每个文件描述符
  - timeout:
    - -1: 永久阻塞
    - 0: 调用完成立即返回
    - >0: 等待的时长毫秒
  - 返回值: IO发送变化的文件描述符的个数

=== 套接字对应的内核缓冲区



在程序中调用write、send往文件描述符对应的缓冲区写数据，数据的发送是有系统完成的

知识点回顾

1. 三次握手
   - 第一次： 客户端 -》 服务器
     - SYN x(0)
   - 第二次：服务器 -》 客户端
     - ACK x+1 SYN y(0)
   - 第三次： 客户端 -》 服务器
     - ACK y+1
2. 四次挥手
   - 端口连接的一方可以是是服务器或者客户端
   - 客户端主动和服务器端口连接
   - 第一次挥手：
     - FIN x+1(0) ACK y+1
   - 第二次挥手：
     - ACK x+2
   - 第三次： 服务器-》客户端
     - FIN y+1(0) ACK x+2
   - 第四次：
     - ACK y+2
3. 并发服务器
   a. 多进程
      i. 父进程：阻塞等待客户端的链接
         1) 如果有连接到达，创建子进程
      ii. 子进程：与客户端进程通信
      iii. 注意事项：
         - 父子进程共享文件描述符
           - 父进程：关闭通信的文件描述符

- 子进程：关闭监听的文件描述符
- 父进程：回收子进程资源
  - 通过信号实现
  - 信号会打断accept的阻塞，accept返回-1
    - 需要errno
- 数据读时共享，写时复制

b. 多线程

i. 父线程：阻塞等待客户端的链接

1) 如果有连接到达，创建子线程

ii. 子线程：通信（客户端）

iii. 注意事项：

1) 父子线程共享堆和全局变量

2) 文件描述符最多1024

3) 数据封装的思想