

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

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);`

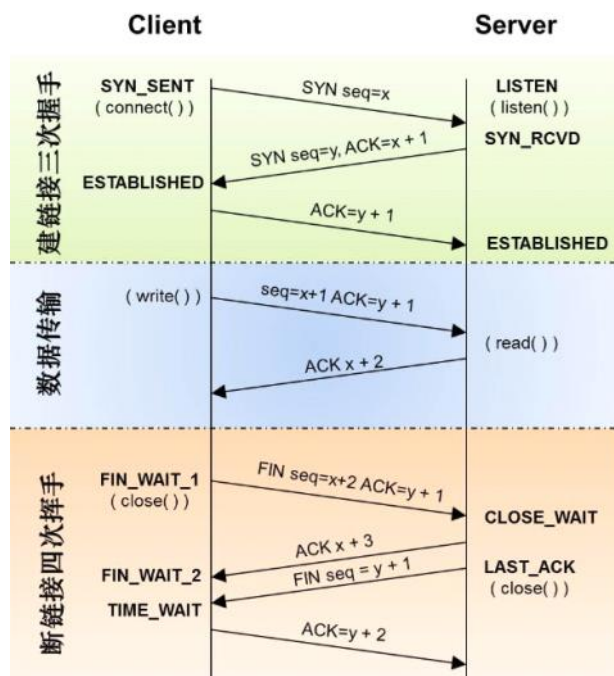
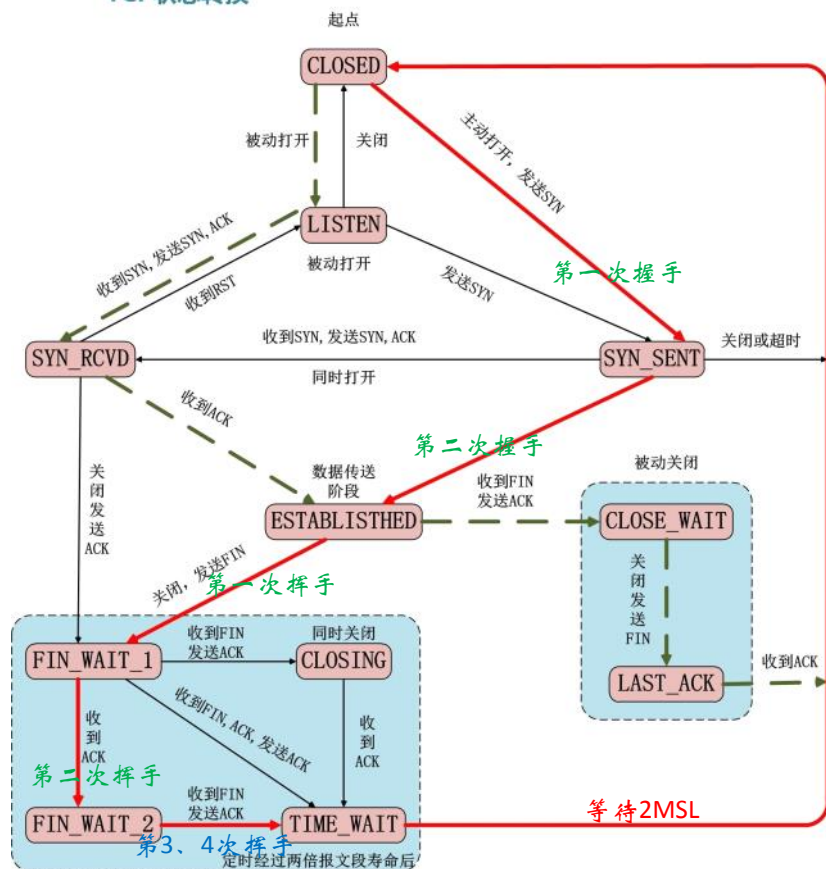
2. 数据发送

- `ssize_t write(int fd, const void *buf, size_t count);`
- `ssize_t send(int sockfd, const void *buf, size_t len, int flags);`

flags 赋值为0

1 - tcp状态转换

TCP状态转换



1. 2MSL

- 等待时长
- 主动关闭连接的一方, 处于TIME_WAIT状态
- 有的地方: 2分钟, 30s, 一般时候是30s(MSL)

2. 半关闭

○ 如何理解?

- A给B发送是FIN(A调用了close函数), 但是B没有给A发送FIN(B没有调用close)
- A断开了与B的连接, B没有断开与A的连接

○ 特点:

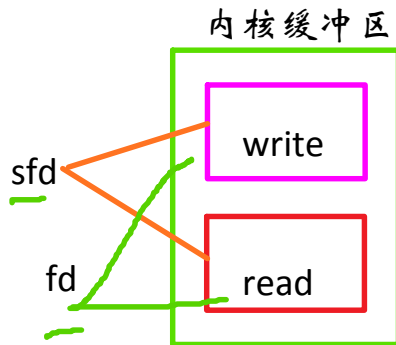
- A不能给B发送数据, A可以收B发送的数据
- B可以给A发送数据

○ 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?



dup2(old, new)

dup2(sfd, fd)

close(fd);

dup

dup2

- 复制文件描述符

1. 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

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:

- 绑定之前设置端口复用的属性

3 - IO多路转接

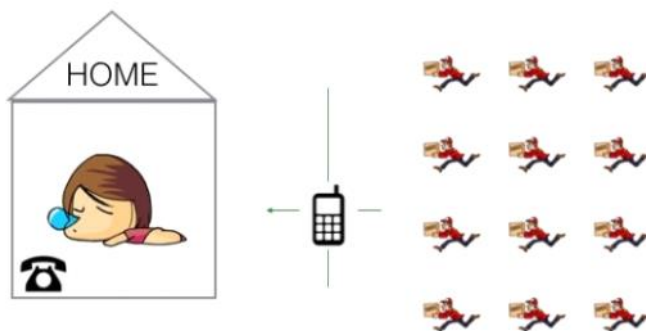
1. IO操作方式

- 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

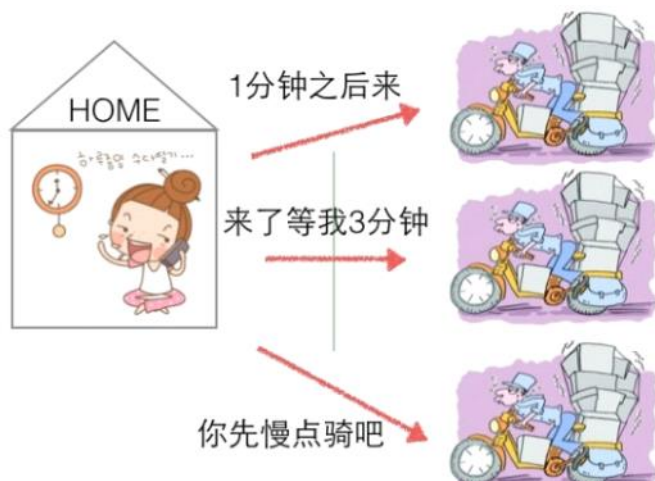
- 优点: 提高了程序的执行效率
- 缺点: 需要占用更多的cpu和系统资源

■ 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



- 解决方案: 使用IO多路转接技术 select/poll/epoll

100

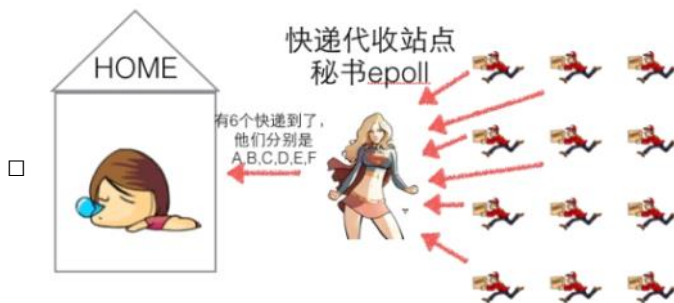
- 第一种: select/poll

30



- 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, 你需要挨个遍历一遍。

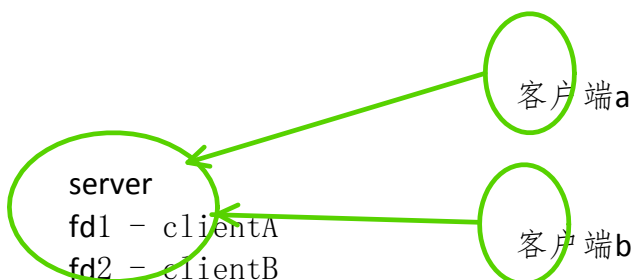
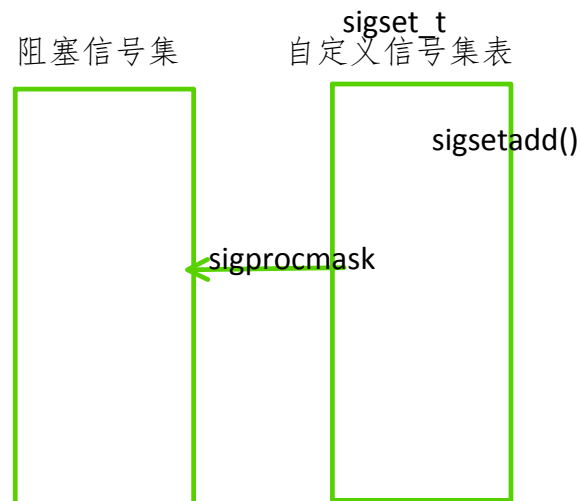
- 第二种: epoll



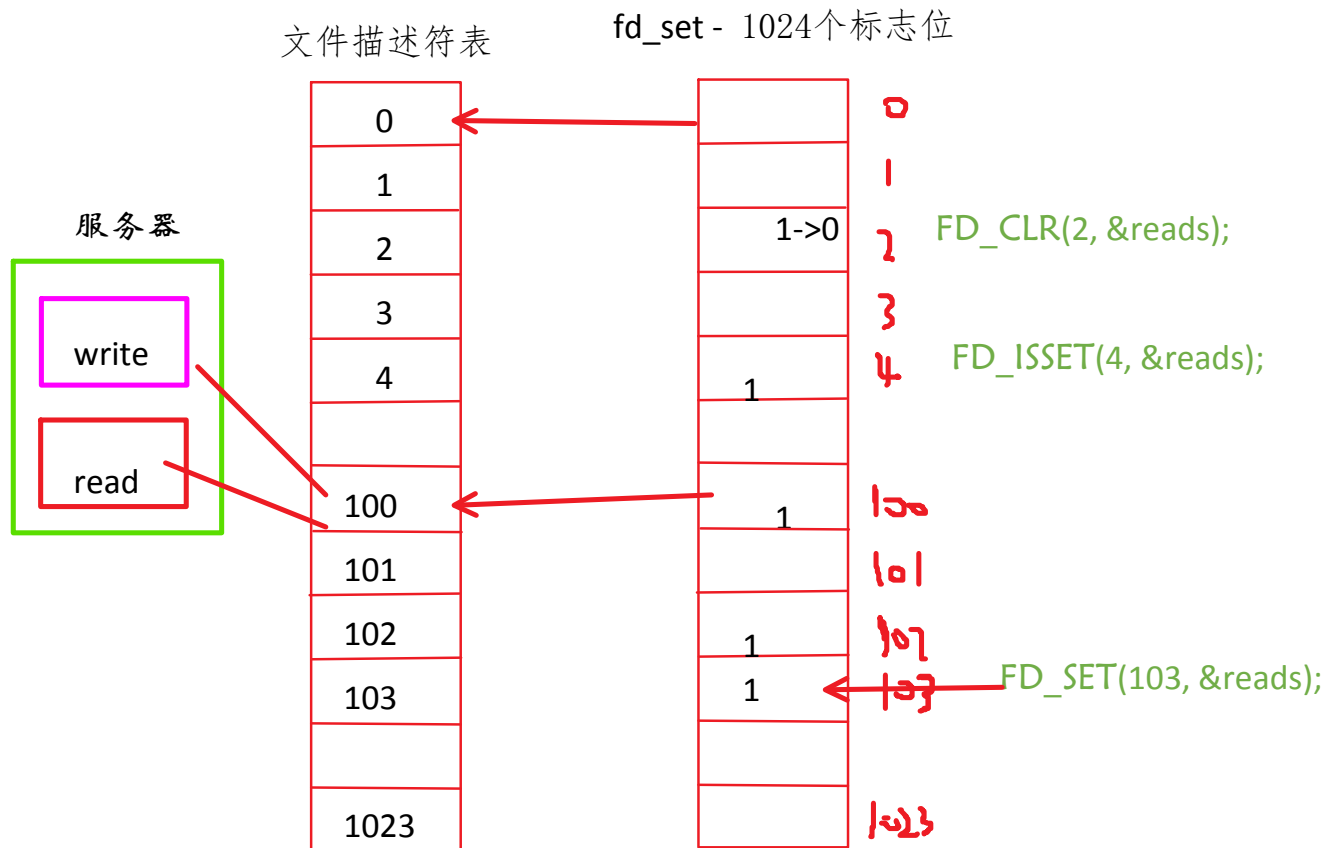
- epoll代收快递员很勤快, 她不仅会告诉你有几个快递到了, 还会告诉你是哪个快递公司的快递。

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



```
settimer()
struct {
    long tv_sec;
    long tv_usec;
};
```

函数原型:

```
int select(int nfd,
```

fd_set *readfds, 传入传出参数

fd_set *writefds,

fd_set *exceptfds,

struct timeval *timeout);

fd

- 读
- 写
- 异常

- 参数:

- o nfd: 要检测的文件描述中最大的fd+1 - 1024
- o readfds: 读集合
- o writefds: 写集合
- o exceptfds: 异常集合
- o timeout:
 - NULL: 永久阻塞
 - 当检测到fd变化的时候返回
 - timeval a;
 - a.tv_sec = 10;
 - a.tv_usec = 0;

- 返回值:

文件描述符集类型: fd_set rdset;

文件描述符操作函数:

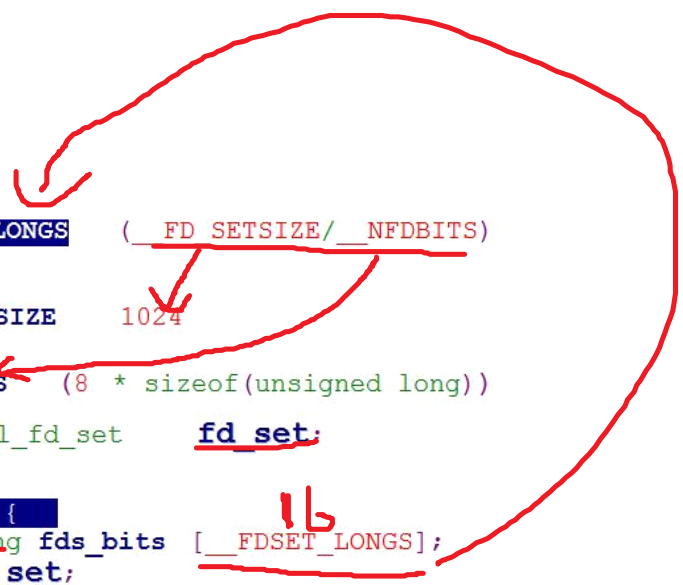
- 全部清空
 - void FD_ZERO(fd_set *set);
- 从集合中删除某一项
 - void FD_CLR(int fd, fd_set *set);
- 将某个文件描述符添加到集合
 - void FD_SET(int fd, fd_set *set);
- 判断某个文件描述符是否在集合中
 - int FD_ISSET(int fd, fd_set *set);

使用select函数的优缺点:

- 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

-----fd_set的内核代码

```
#define FDSET_LONGS    ( __FD_SETSIZE / __NFDBITS )  
  
#define __FD_SETSIZE    1024  
  
#define __NFDBITS    (8 * sizeof(unsigned long))  
  
typedef __kernel_fd_set    fd_set;  
  
typedef struct {  
    unsigned long fds_bits [__FDSET_LONGS];  
} __kernel_fd_set;
```



32x32
11
1524

32-4

64-8

$$1024 / (8 \times 8) = 16 \times 8 \times 8$$
$$4 = 32 \times 4 \times 8$$

---- select工作过程分析

2017年6月1日 23:52

客户端A,B,C,D,E, F连接到服务器
分别对应文件描述符 3, 4, 100, 101, 102, 103

fd_set;	
0	0
1	0
2	0
3	0
4	0
.	0
100	0
101	0
102	0
103	0
	0
1023	0

fd_set;	
0	0
1	0
2	0
3	0
4	0
.	0
100	0
101	0
102	0
103	0
	0
1023	0

5 - IO多路转接 - poll

poll结构体:

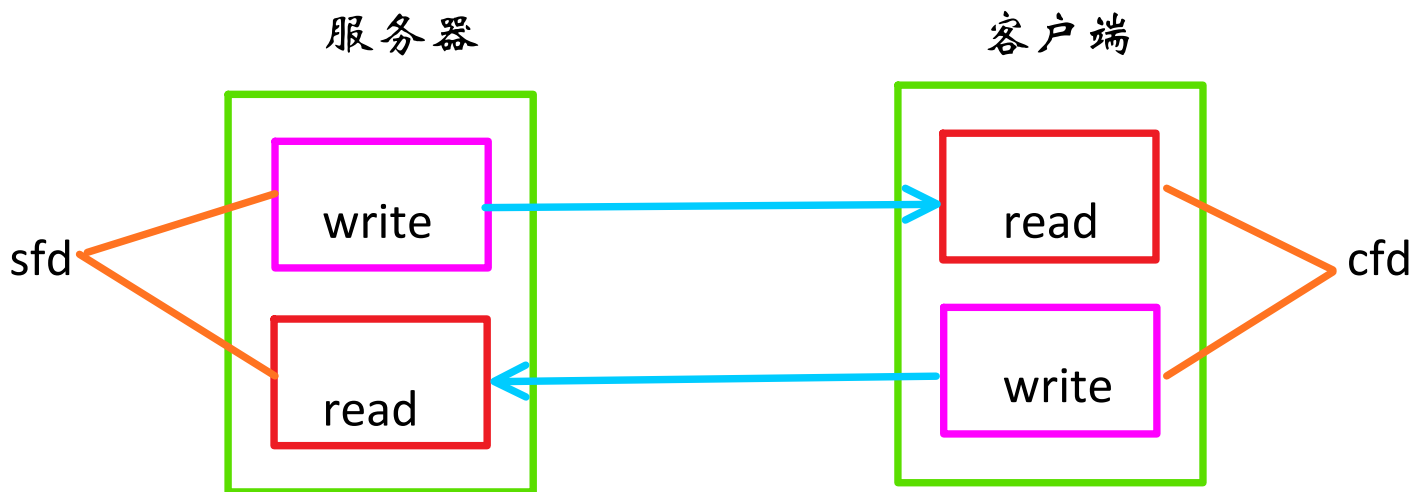
```
struct pollfd {  
    int fd;          /* 文件描述符 */  
    short events;     /* 等待的事件 */  
    short revents;    /* 实际发生的事件 */  
};
```

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



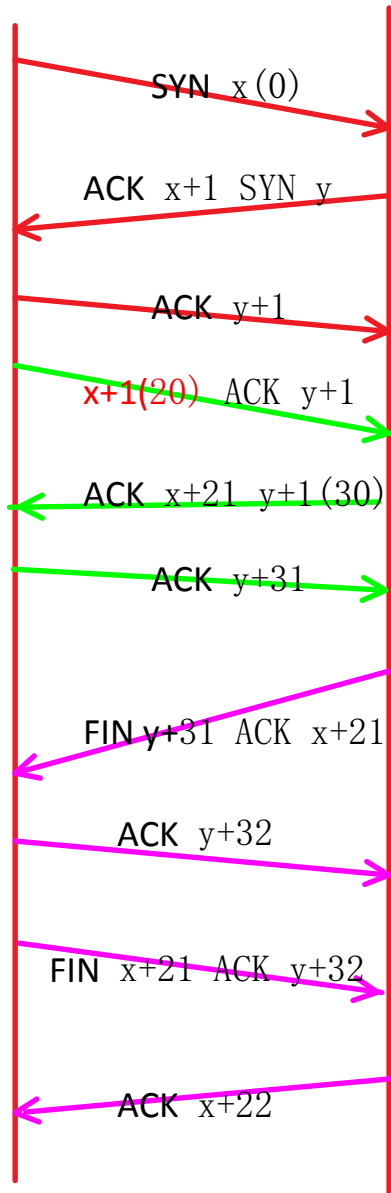
复习

2017年6月2日 9:09

1. 三次握手
2. 四次挥手
3. 并发服务器
 - 进程
 - 线程

客户端

服务器



32位数是多么大?