

学习目标

1. 了解poll操作函数
2. 熟练使用epoll多路IO模型
3. 了解epoll ET/LT触发模式
4. 说出UDP的通信流程

复习

1. tcp状态转换

2. select

○ fd_set - 1024 sizeof(fd_set) = 128

○ select(maxfd+1, &reads, &write, &expt, timeout)

▪ NULL - 永久阻塞

▪ struct timeval val = {1, 0};

▪ reads - 传入传出参数

○ 思路

// 监听

// 将待检测的数据初始化到fd_set集合中

while(1) 循环的委托内核检测

```
{
    select();
    // 判读fd是否是监听的
    // 已经连接的客户端发送来的数据
    for();
    {
        fd_isset(i, &reads);
    }
}
```

FD_ZERO(fd_set*);

FD_SET(fd, fd_set*);

FD_ISSET(fd, fd_set*);

FD_CLR(fd, fd_set*);

128 268 1024

1 - epoll

三个函数:

- 该函数生成一个epoll专用的文件描述符

- `int epoll_create(int size);`

- size: epoll上能关注的最大描述符数

- 用于控制某个epoll文件描述符事件，可以注册、修改、删除

- `int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);`

- 参数:

- epfd: epoll_create生成的epoll专用描述符

- op:

- `EPOLL_CTL_ADD` -- 注册
 - `EPOLL_CTL_MOD` -- 修改
 - `EPOLL_CTL_DEL` -- 删除

- fd: 关联的文件描述符

- event: 告诉内核要监听什么事件

- 等待IO事件发生 - 可以设置阻塞的函数

- `int epoll_wait(int epfd, struct epoll_event* events, // 数组, int maxevents, int timeout`

- `);`

- 对应select和poll函数

- 参数:

- epfd: 要检测的句柄
 - events: 用于回传待处理事件的数组
 - maxevents: 告诉内核这个events的大小
 - timeout: 为超时时间
 - -1: 永久阻塞
 - 0: 立即返回
 - >0:

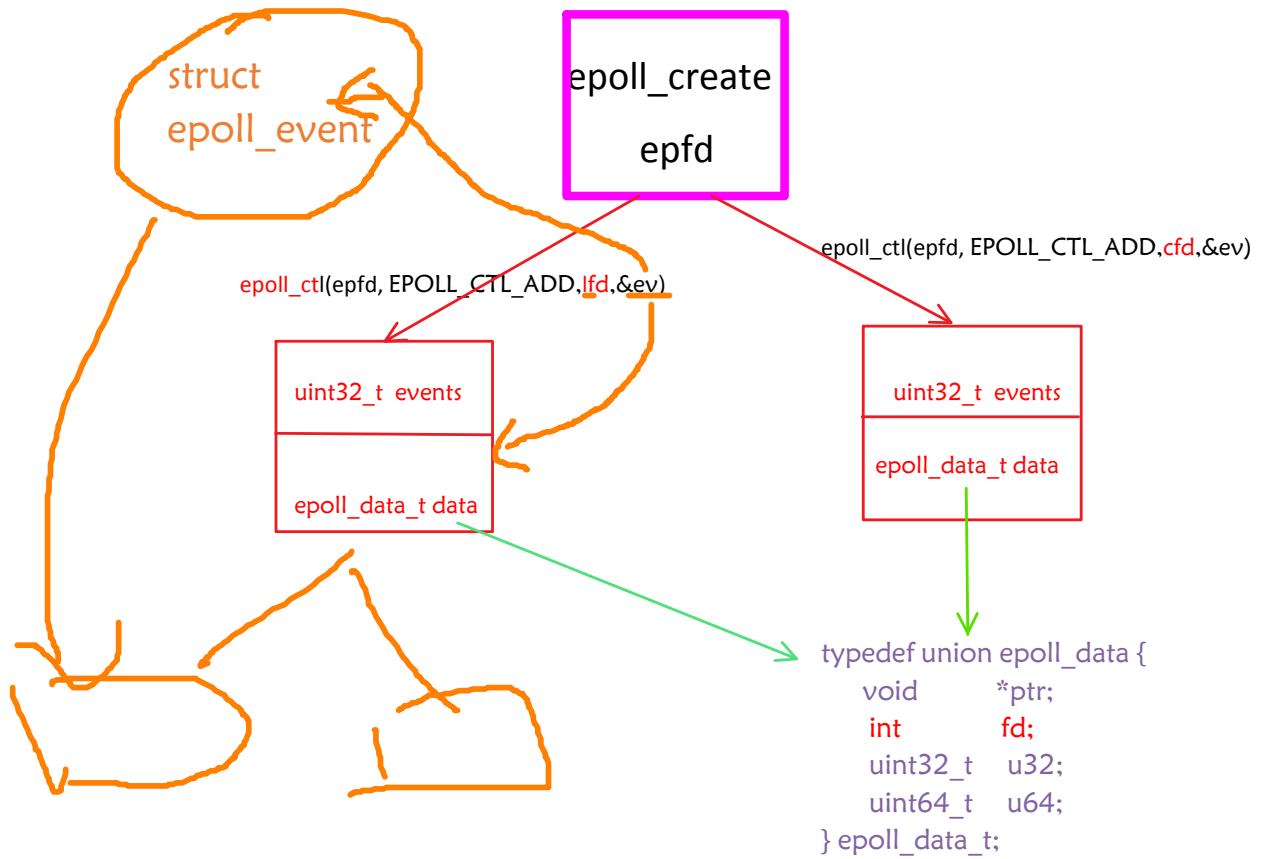
```
typedef union epoll_data {  
    void *ptr;  
    int fd;  
    uint32_t u32;  
    uint64_t u64;  
} epoll_data_t;
```

```
struct epoll_event {  
    uint32_t events;  
    epoll_data_t data;  
};
```

events:

- `EPOLLIN` - 读
- `EPOLLOUT` - 写
- `EPOLLERR` - 异常

- 1-1- epoll树



- 1-2 epoll模型

2017年6月4日 10:27

```
int main()
{
    // 创建监听的套接字
    int lfd = socket();
    // 绑定
    bind();
    // 监听
    listen();

    // epoll树根节点
    int epfd = epoll_create(3000);
    // 存储发送变化的fd对应信息
    struct epoll_event all[3000];
    // init
    // 监听的lfd挂到epoll树上
    struct epoll_event ev;
    // 在ev中init lfd信息
    ev.events = EPOLLIN;
    ev.data.fd = lfd;
    epoll_ctl(epfd, EPOLL_CTL_ADD, lfd, &ev);
    while(1)
    {
        // 委托内核检测事件
        int ret = epoll_wait(epfd, all, 3000, -1);
        // 根据ret遍历all数组
        for(int i=0; i<ret; ++i)
```

```

{
    int fd = all[i].data.fd;
    // 有新的连接
    if( fd == lfd)
    {
        // 接收连接请求 - accept不阻塞
        int cfd = accept();
        // cfd上树
        ev.events = EPOLLIN;
        ev.data.fd = cfd;
        epoll_ctl(epfd, epoll_ctl_add, cfd,
            &ev);
    }
    // 已经连接的客户端有数据发送过来
    else
    {
        // 只处理客户端发来的数据
        if(!all[i].events & EPOLLIN)
        {
            continue;
        }
        // 读数据
        int len = recv();
        if(len == 0)
        {
            close(fd);
            // 检测的fd从树上删除
            epoll_ctl(epfd, epoll_ctl_del, fd,
                NULL);
        }
    }
}

```

```
        // 写数据  
        send();  
    }  
}  
}
```

2 - 文件描述符突破1024限制

1. select - 突破不了, 需要编译内核
2. poll和epoll可以突破1024限制

1. 查看受计算机硬件限制的文件描述符上限
 -
2. 通过配置文件修改上限值

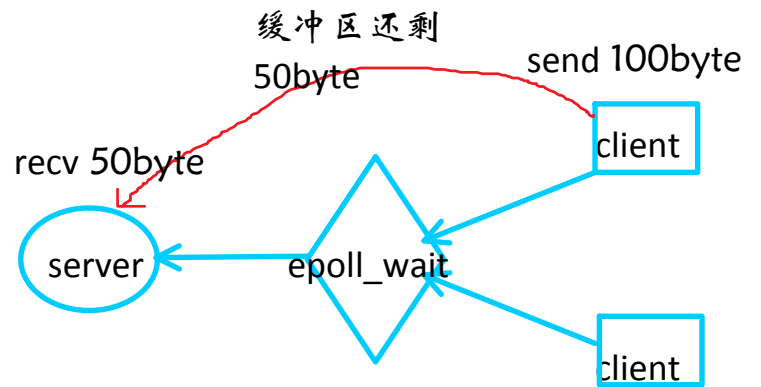
3 - epoll 三种工作模式

1. 水平触发模式

○

2. 边沿触发模式

3. 边沿非阻塞触发



epoll_wait 调用次数越多, 系统的开销越大

fcntl

- 文件打开之后修改文件属性
- 可以获胜设置的属性 flags
- 获取flags
 - `int flags = fcntl(fd, F_GETFL);`
- 设置flags
 - `flags = flags | O_NONBLOCK;`
 - `fcntl(fd, F_SETFL, flags);`

4 - 心跳包

1. 判断客户端和服务端是否处于连接状态

○ 心跳机制

- 不会携带大量的数据
- 每个一定时间服务器->客户端/客户端->服务器发送一个数据包

○ 心跳包看成一个协议

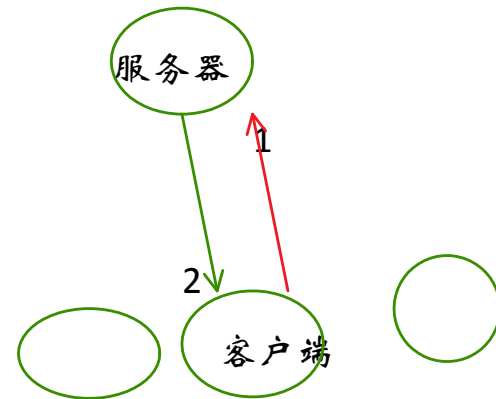
- 应用层协议

○ 判断网络是否断开

- 有多个连续的心跳包没收到/没有回复
- 关闭通信的套接字

○ 重连

- 重新初始套接字
- 继续发送心跳包



--乒乓包

- 比心跳包携带的数据多一些
- 除了知道连接是否存在, 还能获取一些信息

5 - UDP通信

1. tcp - 面向连接的的安全的数据包通信

a. 基于流 sock_stream

2. udp - 面向无连接不安全报文传输

3. 通信流程

○ 服务器端:

■ 创建套接字 - socket

□ 第二个参数: SOCK_DGRAM

■ 绑定IP和端口: bind

□ fd

□ struct sockaddr -- 服务器

■ 通信

□ 接收数据: recvfrom

ssize_t **recvfrom**(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen);

◆ fd - 文件描述符

◆ buf: 接收数据缓冲区

◆ len: buf的最大容量

◆ flags: 0

◆ src_addr: 另一端的IP和端口, 传出参数

◆ addrlen: 传入传出参数

□ 发送数据: sendto

ssize_t **sendto**(int sockfd, const void

*buf, size_t len, int flags, const struct
sockaddr *dest_addr, socklen_t
addrlen);

- ◆ sockfd: socket函数创建出来的
- ◆ buf: 存储发送的数据
- ◆ len: 发送的数据的长度
- ◆ flags: 0
- ◆ dest_addr: 另一端的IP和端口
- ◆ addrlen: dest_addr长度

■ udp服务器端: 需要一个套接字, 通信

○ 客户端:

- 创建一个用于通信的套接字: socket
- 通信

□ 发送数据: sendto

- ◆ 需要先准备好一个结构体: struct
sockaddr_in

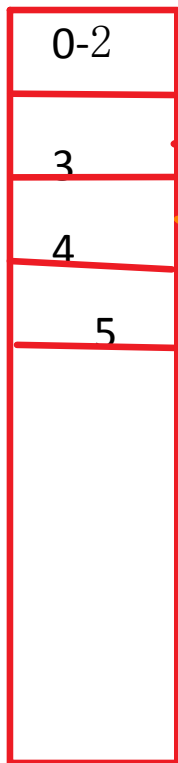
◇ 存储服务器的IP和端口

□ 接收数据: recvfrom

○ udp的数据是不安全的, 容易丢包

- 丢包, 丢全部还一部分?
 - 只能丢全部
- 优点: 效率高

文件描述符表



创建一个树节点

?

fd

fd

```
struct epoll_event ev;
```