Linux 网络编程阶段

- 第一天
  - 网络基础
    - 概念和相关协议介绍
  - socket编程
- 第二天
  - tcp三次握手
  - tcp四次挥手
  - 并发
    - 多进程
    - 多线程
- 第三天
  - tcp的状态转换
  - select
  - poll
- 第四天
  - epoll
    - 水平
    - 边沿
    - 边沿非阻塞
  - udp通信
- 第五天
  - 广播
  - 组播
  - 本地套接字
- 第六天
  - libevent开源库

第七天

- Json
- Xml

第八，九, 十天

- 自己实现一个Mini Web服务器

http协议 b-s

1 - 网络基础

1. 网络应用程序设计模式:
   - C/S - client/server
     - 优点: 1. 协议选用灵活 2. 可以缓存数据
     - 缺点: 1. 对用户安全构成威胁 2. 开发工作量大, 调试困难
   - B/S - browser/server
     - 优点: 跨平台
     - 缺点: 只能使用http
2. 协议的概念
   - 规则: 数据传输和数据解释的规则
   - 原始协议 ------>(改进、完善)------> 标准协议
   - 典型协议: TCP/UDP HTTP FTP IP ARP
3. 分层模型
   - 7层模型 - OSI:

     物 -- 双绞线, 光纤

     数 -- 数据的传输和错误检测

     网 -- 为数据包选择路由

     传 -- 提供端对端的接口 tcp/udp

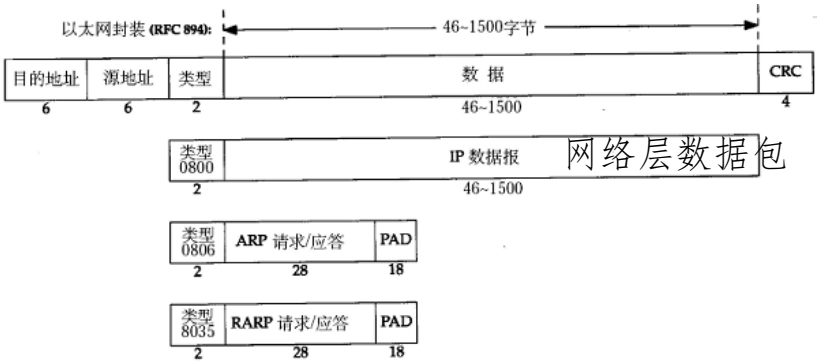
     会 -- 解除或建立与别的节点的联系

     表 -- 数据格式化,代码转换,数据加密

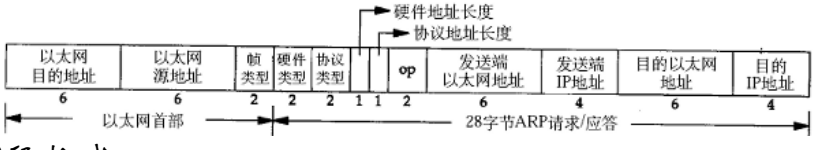
     应 -- 文件传输,电子邮件,文件服务,虚拟终端
   - 4层模型 - TCP/IP: 网络接口层 网络层 传输层 应用层
4. 协议格式

   -- 数据包的封装思想

   1>. 以太网帧格式 -- 借助mac地址完成数据报传递



   □ arp数据报 -- 根据IP获取mac地址



   2>. IP段格式:

   4位版本: ipv4 ipv6

8位生存时间(TTL): 最多能经过多少跳

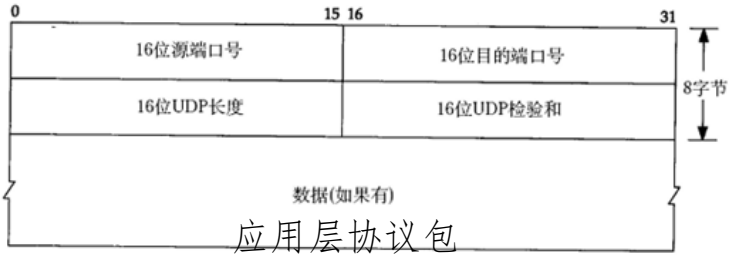
32位源IP地址: 数据发送端地址

32位目的IP地址: 数据接收端地址

3>. UDP数据包格式

16位源端口:

16位目的端口:



4>. TCP数据报格式

16位源端口

16位目的端口

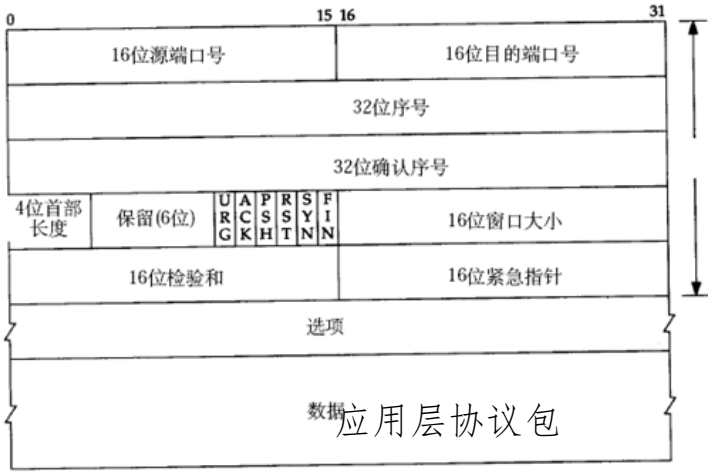
32位序号

32位确认序号

6个标志位

16位滑动窗口

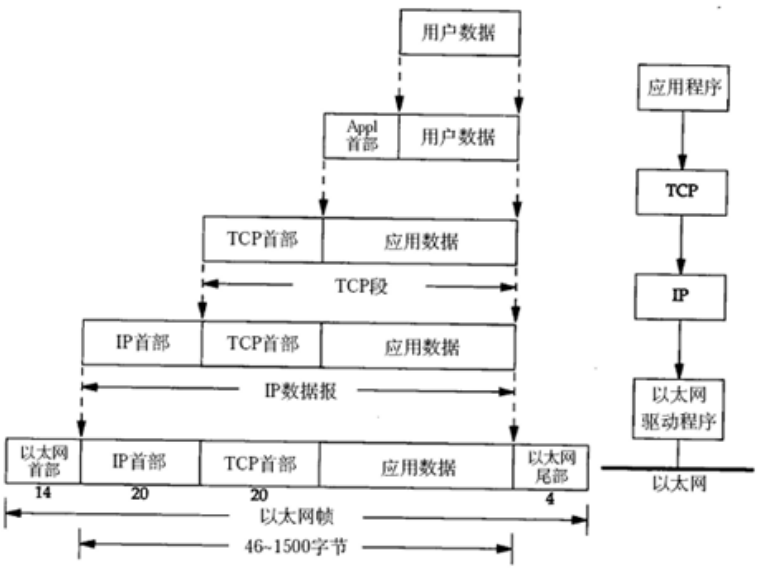
◆ 存储空间



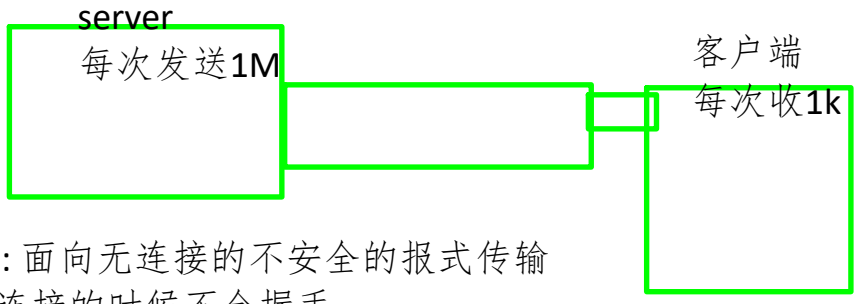
5. 数据的发送和接收

进程

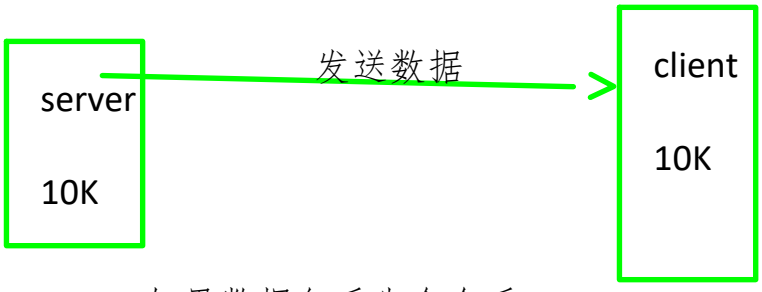
- 进程ID
- 网络环境中
  - IP - 定位一台主机
  - Port - 定位一个进程
  - 127.0.0.1:80
- 端口: 16位
  - 2的16次方
  - 65535



6. tcp, udp传输层协议
   - tcp: 面向连接的安全的流式传输协议
     - 连接的时候, 进行三次握手
     - 数据发送的时候, 会进行数据确认
       - 数据丢失之后, 会进行数据重传



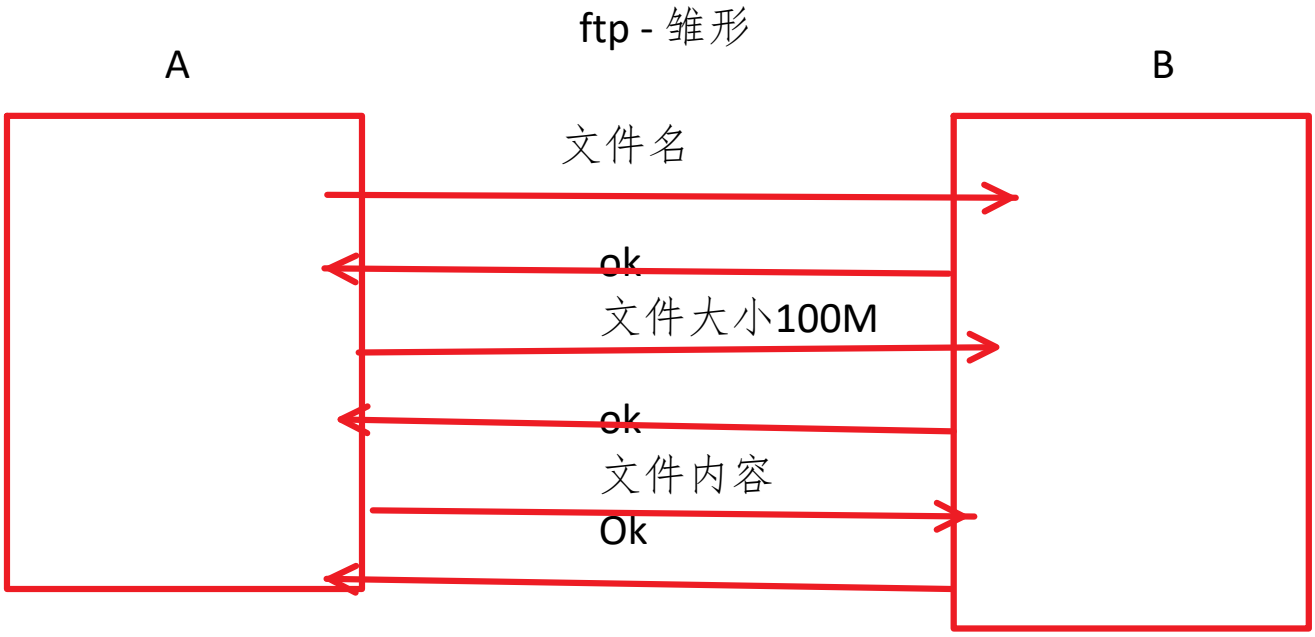
   - udp: 面向无连接的不安全的报式传输
     - 连接的时候不会握手
     - 数据发送出去 之后就不管了



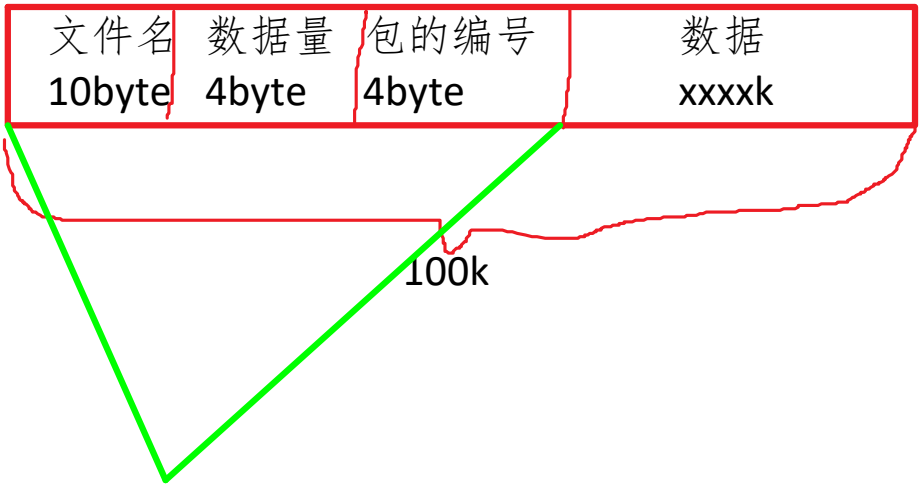
如果数据包丢失会全丢
不存在丢失一半的情况

协议

2017年5月31日 9:22



协议
数据包



协议头

2 - socket编程

socket编程

- 什么是socket
  - 网络通信的函数接口
  - 封装了传输层协议
    - tcp
    - udp
- 浏览器 - http
  - 封装的是tcp

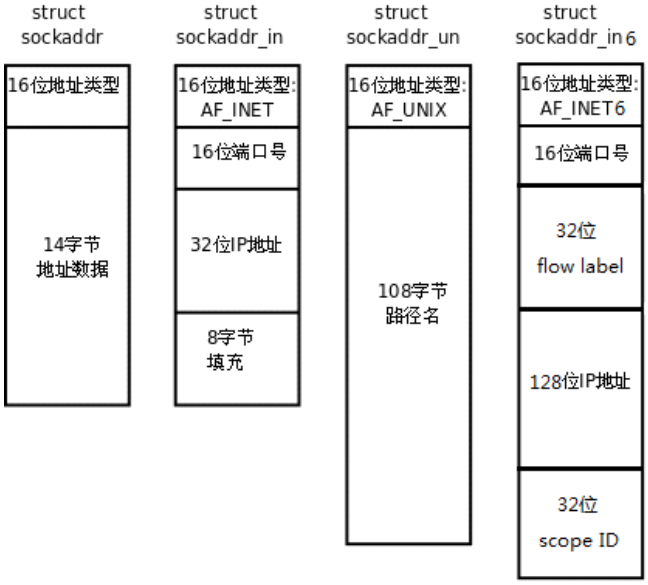
1. 套接字概念
   - IP地址:
   - 端口号:
   - IP+Port:
2. 网络字节序
   - 大端: - 网络字节序
     - 数据的高位字节 - 存储在内存的低地址位
   - 小端: - 主机字节序
     - 数据的高位字节 - 存储在内存的高地址位
     - 常见主机数据是小端存储
   - 作业: 写程序验证数据是大端存储还是小端存储?
   - 相关函数 --
     - 头文件: #include <arpa/inet.h>
     - 类型: int -> int
     - 主机字节顺序 --> 网络字节顺序
uint16_t htons(uint16_t hostshort);	端口
uint32_t htonl(uint32_t hostlong);	IP
     - 网络字节顺序 --> 主机字节顺序
uint16_t ntohs(uint16_t netshort);	端口
uint32_t ntohl(uint32_t netlong);	IP
3. IP地址转换函数

a. 指定IP - 字符串(点分十进制)

- 本地IP转网络字节序　字符串 --> int(大端方式存储)
  - int inet_pton(int af, const char *src, void *dst);
  - 参数:
    - af

        - src
        - dest
    - 网络字节序转本地IP int -> 字符串
        - const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
        - 参数:
            - af
                - af_inet
            - src
                - 网络字节序的整形IP
            - dst
            - size

4. sockaddr数据结构
    - sockaddr
    - sockaddrin
    - sockaddrun
    -




```
struct sockaddr {
    /* address family, AF_xxx */
    sa_family_t sa_family;
    /* 14 bytes of protocol address */
    char sa_data[14];
};
struct sockaddr_in {
    __kernel_sa_family_t sin_family;  // 地址族协议
    __be16 sin_port;  // 端口
    struct in_addr sin_addr;  // IP地址
    unsigned char __pad[__SOCK_SIZE__ - sizeof(short int) -
    sizeof(unsigned short int) - sizeof(struct in_addr)];
};
struct in_addr {
    __be32 s_addr;
};
```

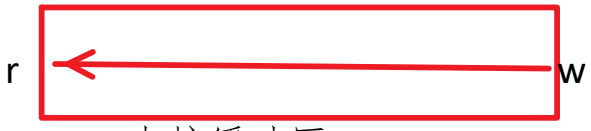
5. 网络套接字函数
   - int socket(int domain, int type, int protocol);
     - 创建套接字
       - domain
         - ipv4 af_inet
       - type
         - tcp - 流式协议
         - udp - 报式协议
       - protocol - 0
       - 返回值:文件描述符(套接字)
   - int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
     - 将本地的IP和端口与创建出的套接字绑定
     - 参数:
       - sockfd - 创建出的文件描述符
       - addr - - 端口和IP
       - addrlen - addr结构体的长度
   - int listen(int sockfd, int backlog);
     - 设置同时连接到服务器的客户端的个数
     - 参数:
       - socket函数创建出来的文件描述符
       - backlog - -最大值 128
   - int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
     - 阻塞函数
     - 阻塞等待客户端连接请求, 并接受连接
     - 参数:
       - sockfd:文件描述符, 使用socket创建出的文件描述符
         - 监听的文件描述符
       - addr: 存储客户端的端口和IP, 传出参数
       - addrlen: 传入传出参数
     - 返回值: 返回的是一个套接字, 对应客户端
       - 服务器端与客户端进程通信使用accept的返回值对应的套接字
   - int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
     - sockfd: 套接字
     - addr: 服务器端的IP和端口
     - addrlen: 第二个参数的长度
6. C/S模型 - TCP -- 面向连接的可靠数据包传递
   - 服务器端:

- 客户端

---- 套接字 - socket

socket编程 - 网络IO编程
- 读写操作
- read/wirte
  - 文件描述符
- 创建一个套接字,得到是文件描述符

管道: - 匿名

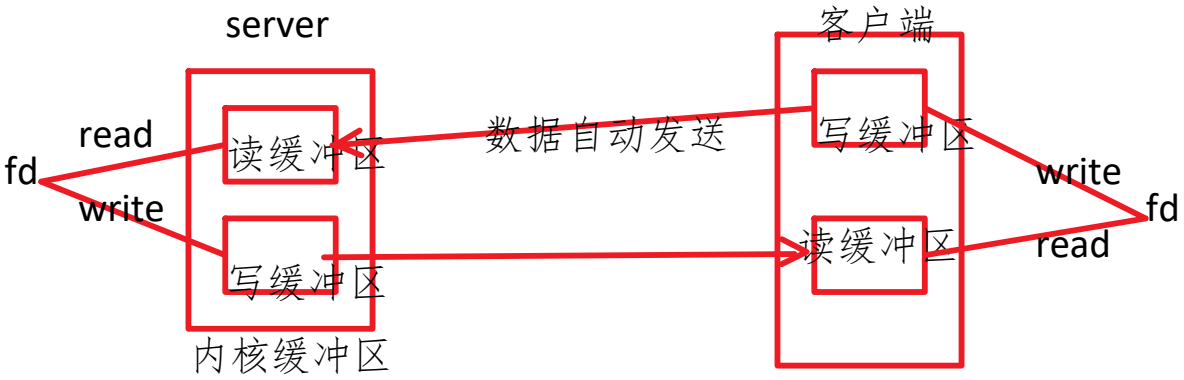


内核缓冲区
内存中一块存储空间
管道的读写两端分别对应一个文件描述符

套接字 -
- 创建成功,得到一个文件描述符 fd
- fd操作的是一块内核缓冲区

server

客户端


server
客户端
read
fd
write
读缓冲区
数据自动发送
写缓冲区
write
fd
read
写缓冲区
读缓冲区
内核缓冲区

默认也是阻塞的

-----大端-小端

整数: 0x12345678 - 4字节

大端法表示

内存地址	0x4000	0x4001	0x4002	0x4003
存放内容	0x12	0x34	0x56	0x78

小端法表示

内存地址	0x4000	0x4001	0x4002	0x4003
存放内容	0x78	0x56	0x34	0x12

常见文件字节序：

Adobe PS – Big Endian

BMP – Little Endian

GIF – Little Endian

JPEG – Big Endian

MacPaint – Big Endian

RTF – Little Endian

另外，Java和所有的网络通讯协议都是使用Big-Endian的编码

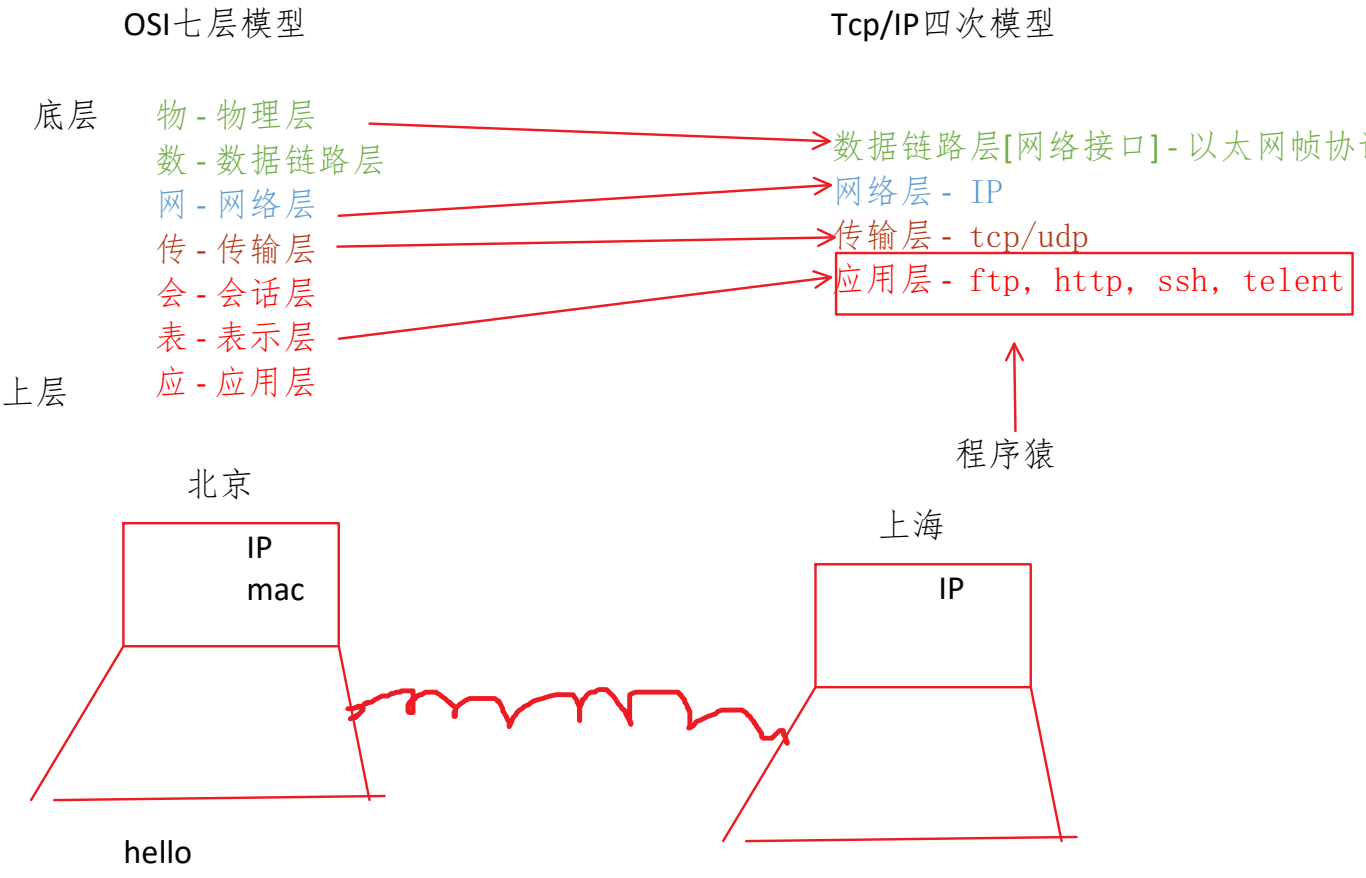
oxff
11111111

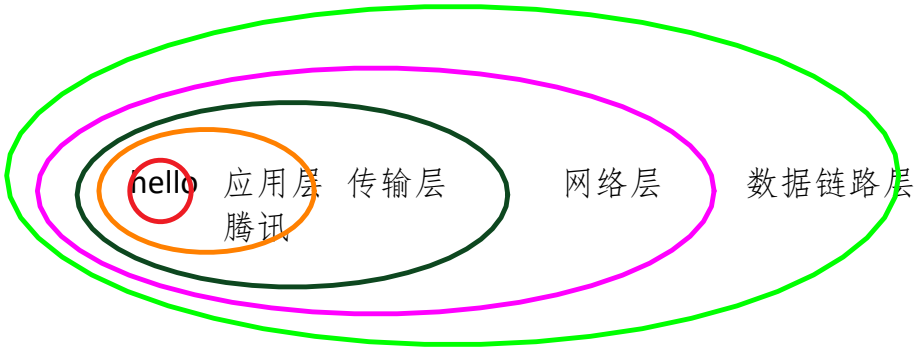
100000000

1. 思考题:
   写程序测试当前电脑数据的存储方式?
   - int a = 0x1234;
   - 从内存的地位取一个字节
   - 读这个字节中的数

网络模型

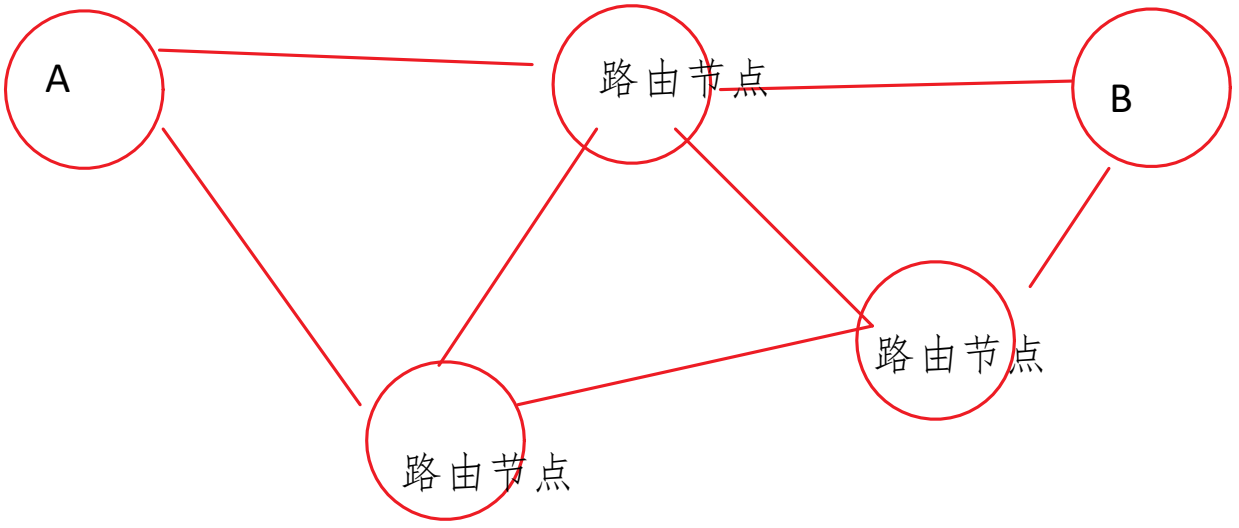
2017年5月31日 9:42





2017年5月31日 9:45

网络



DNS

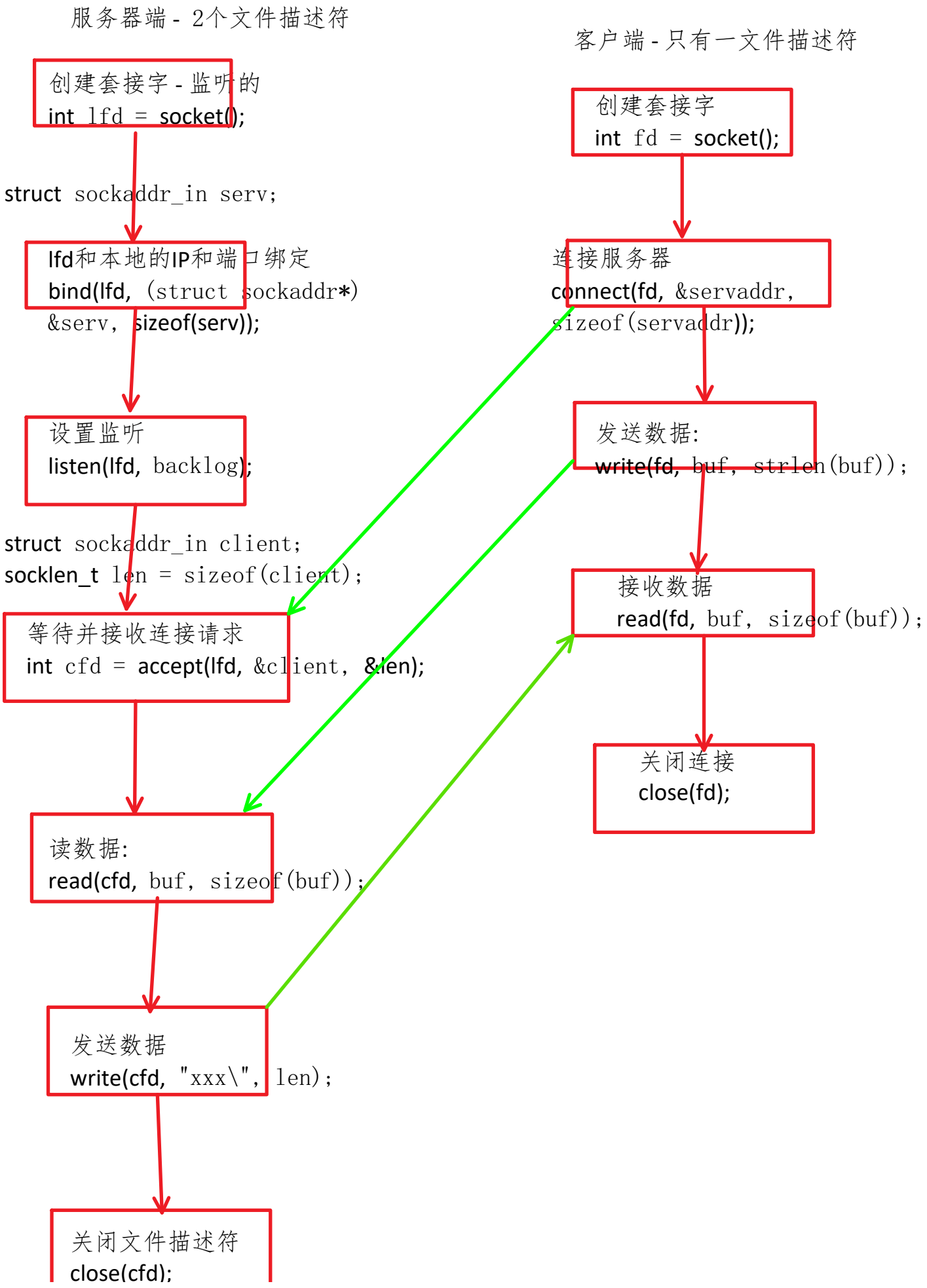
2017年5月31日 10:42

DNS - 服务器

- 域名解析服务器
- www.baidu.com
  - 拿到了IP地址
  - DNS根服务器
    - 13台
      - 10 - 老美
      - 1 - 英国
      - 1 -瑞典
      - 1 - 鬼子

tcp通信流程

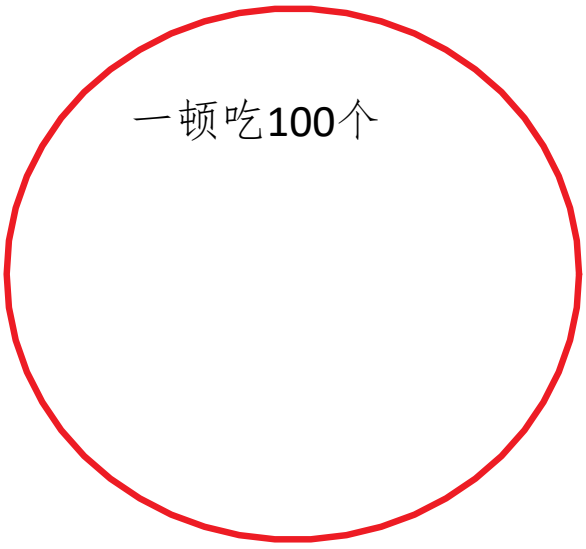
2017年5月31日 15:12

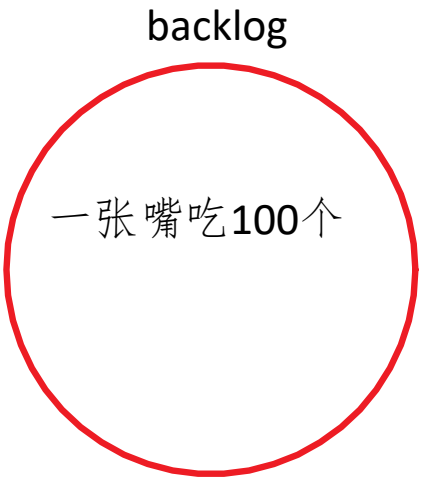


关闭文件描述符
close(cfd);
close(lfd);

服务器端 - 2个文件描述符
- 监听 - socket
- 通信 - accept的返回值
  - read
  - write

2017年5月31日 15:30





2017年5月31日 16:19

192.168.1.100 - 字符串