# 第11章 网络编程

## 11.1 客户端-服务器编程模型

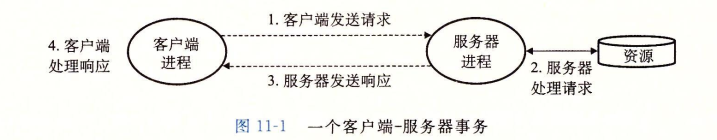
客户端服务器的基本操作是事务(transaction), 一个客户端服务器事务的步骤:

1) 客户端发送请求

2) 服务器处理请求

3) 服务器发送响应

4) 客户端处理响应



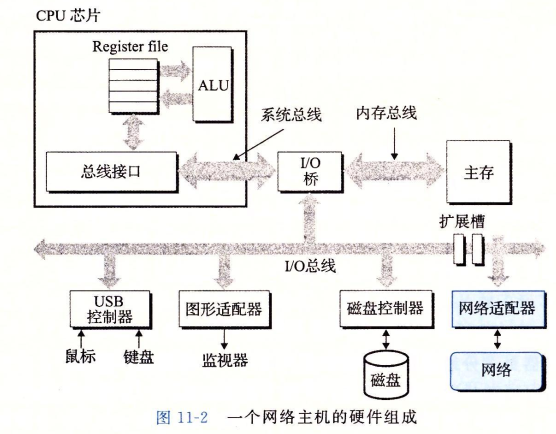
客户端和服务器是**进程**, 而不是主机.

## 11.2 网络

对于主机而言, 网络又是一种IO设备.

一个网络主机的硬件组成:

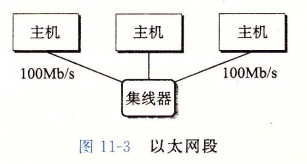
CPU芯片 系统总线 IO桥 内存总线 主存 IO总线 各种适配器和控制器



局域网(LAN, Local Area Network)是最底层的网络.

最流行的局域网技术是以太网(Ethernet). 以太网于1970s提出, 从3Mb/s演变到10Gb/s.

一个以太网段包含一些电缆(通常为双绞线)和一个集线器.



集线器将信号发送给所有连接在它上面的线路.

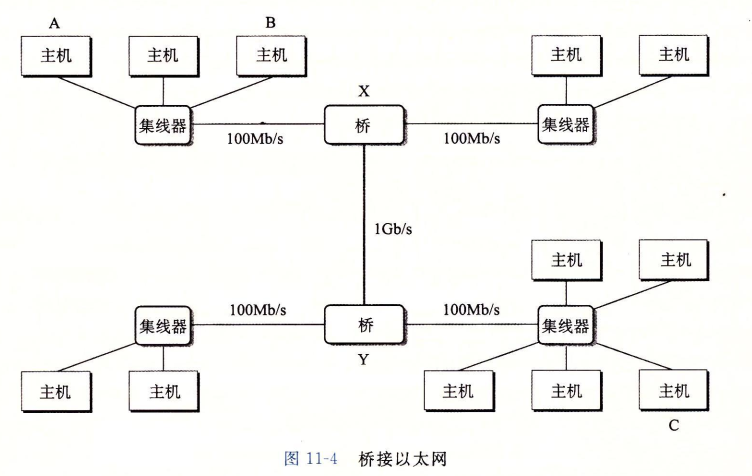
扩展:

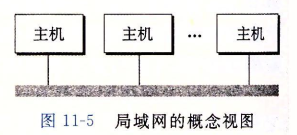
交换机根据MAC地址表进行包转发操作.

每个以太网适配器都有一个全球唯一的48位地址, 存储在非易失性存储器上.

桥接以太网: 使用一些电缆和网桥(交换机)可以将多个以太网段连接成较大的局域网.

互联网络: 多个不兼容的局域网通过路由器连接起来, 形成互联网络.





## 11.3 全球IP因特网

IP地址结构

|  |
| --- |
| struct in\_addr {  uint32\_t s\_addr;  }; |

注: 将IP地址存放在一个结构体之中是早期实现的缺陷.

网络字节序是大端序.

UNIX提供以下函数再网络和主机字节序之间实现转化:

|  |
| --- |
| #include <arpa/inet.h>  uint32\_t htonl(uint32\_t hostlong);  uint32\_t htons(uint16\_t hostshort);  uint32\_t ntohl(unit32\_t netlong);  uint32\_t ntohs(unit16\_t netshort); |

n: network data

h: host data

l: 32位无符号整型数字

s: 16位无符号整型数字

在LINUX系统上查看主机的点分十进制IP地址:

$ hostname -i

172.20.153.170

实现点分十进制和二进制IP地址的转换:

|  |
| --- |
| #include <arps/inet.h>  int inet\_pton(AF\_INET, const char \*src, void \*dst);  成功返回1, 若src非法返回0, 出错返回1.  const char \*inet\_ntop(AF\_INET, const void \*src, char \*dst, socklen\_t size);  成功返回指向点分十进制字符串的指针, 出错返回NULL. |

域名系统(DNS, Domain Name System): 定义了一组域名和一组IP地址之间的映射.

LINUX可以通过nslookup命令查看IP地址和域名之间的映射

|  |
| --- |
| **$ nslookup localhost**  Server: 127.0.0.53  Address: 127.0.0.53#53  **$ nslookup www.baidu.com**  Server: 127.0.0.53  Address: 127.0.0.53#53  Non-authoritative answer:  www.baidu.com canonical name = www.a.shifen.com.  Name: www.a.shifen.com  Address: 220.181.38.150  Name: www.a.shifen.com  Address: 220.181.38.149 |

域名和IP地址不必一一映射.

套接字地址格式

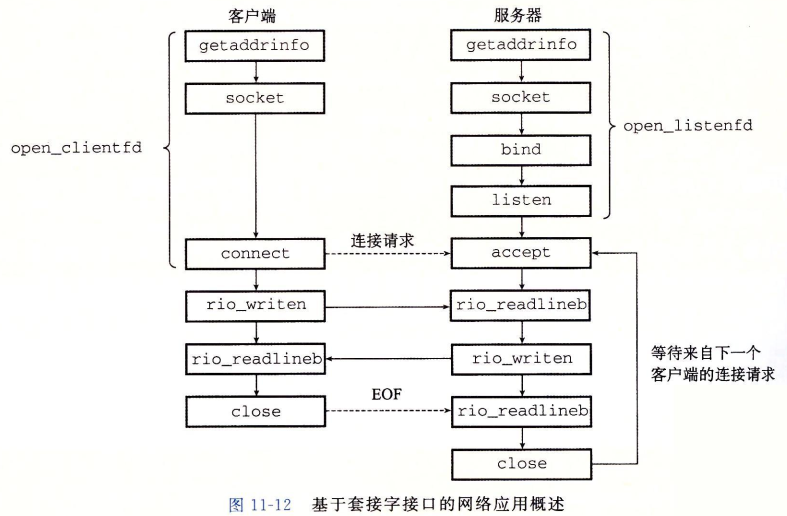
(地址:端口)

一对连接的表示

(cliaddr:cliport servaddr:servport)

文件/etc/services包含了主机上知名名字和知名端口之间的映射.

## 11.4 套接字接口



套接字接口是一组函数, 它们和UNIX IO函数结合起来, 用以创建网络应用.

套接字地址结构

|  |
| --- |
| /\* IP套接字地址结构 \*/  struct sockaddr\_in {  uint16\_t sin\_family; /\* 协议栈(总是AF\_INET) \*/  uint16\_t sin\_port; /\* 网络字节序的端口号 \*/  struct in\_addr sin\_addr; /\* 网络字节序的IP地址 \*/  unsigned char sin\_zero[8]; /\* \*/  };  /\* 通用的套接字地址结构(用于connect, bind和accept) \*/  struct sockaddr {  uint16\_t sa\_family; /\* 协议栈(总是AF\_INET) \*/  char sa\_data[14];  }; |

struct sockaddr存在的意义:

这个结构体是专门为connect, bind和accept设计, 这三个函数要求一个指向与协议相关的套接字地址结构的指针. 当时设计套接字接口的时候C语言还没有void\*指针, 所以设计者设定套接字函数要求指向一个通用的sockaddr结构的指针, 然后要求应用程序应用程序将与协议特定的结构的指针强制转换成这个通用结构.

客户端和服务器使用socket函数来创建一个套接字描述符.

|  |
| --- |
| #include <sys/types.h> /\* 其实LINUX不需要包含这个文件, 只是为了兼容 \*/  #include <sys/socket.h>  int socket(int domain, int type, int protocol);  成功返回非负描述符, 若出错返回-1. |

socket返回的套接字是部分打开的, 还不能用于读写.

客户端通过connect函数来建立和服务器的连接.

|  |
| --- |
| #include <sys/socket.h>  int connect(int clientfd, const struct sockaddr \*addr, socklen\_t addrlen);  成功返回0, 出错返回-1. |

connect函数试图与套接字地址为addr的服务器建立一个因特网连接, 其中addrlen为sizeof(sockaddr\_in).

connect函数会阻塞.

bind函数告诉内核将addr中的服务器套接字地址和套接字描述符sockfd起来.

|  |
| --- |
| #include <sys/socket.h>  int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);  成功返回0, 出错返回-1. |

客户端是发起连接的主动实体. 服务器是来自客户端的连接请求的被动实体. 默认情况下, 内核会认为**socket**函数创建的描述符对应于**主动套接字**, 它存在于一个连接的客户端. 服务器调用listen函数告诉内核, 描述符是被服务器而不是客户端使用.

|  |
| --- |
| #include <sys/socket.h>  int listen(int sockfd, int backlog);  成功返回0, 出错返回-1. |

listen函数将sockfd从一个主动套接字转化为一个监听套接字, 该套接字可以接收来自客户端的请求.

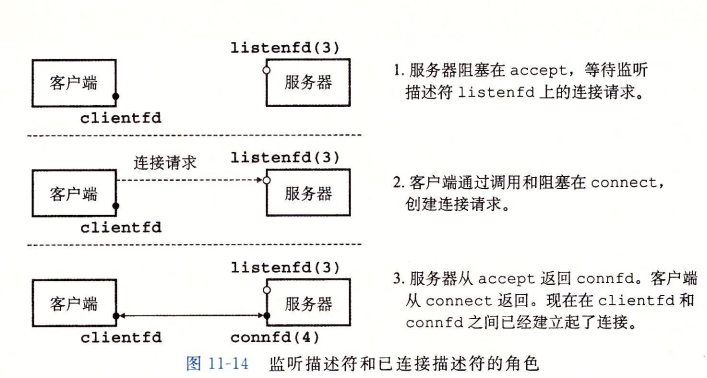
backlog参数表示连接请求队列的最大长度, 超过这个长度内核将拒绝连接请求, 一般设定为1024即可.

服务器通过调用accept函数来等待来自客户端的连接请求.

|  |
| --- |
| #include <sys/socket.h>  int accrpt(int listenfd, struct sockaddr \*addr, int \*addrlen);  成功返回非负描述符, 失败返回-1. |

accept等待要连接到listenfd的连接请求, 然后在addr中填写客户端的套接字地址, 并返回一个已连接描述符, 这个描述符可以被用来与客户端通信.

listenfd叫做监听描述符.



区别监听描述符和已连接描述符使得我们可以建立并发服务器, 它能够同时处理许多客户端连接.

getaddrinfo()函数将主机名, 主机地址, 服务器名和端口号的字符串转化成套接字地址结构.

getaddrinfo()是gethostbyname()和getservbyname()函数的替代品.

getaddrinfo()可重入, gethostbyname()和getservbyname()不可重入.

|  |
| --- |
| #include <sys/types.h>  #include <sys/socket.h>  #include <netdb.h>  int getaddrinfo(const char \*host,  const char \*service,  const struct addrinfo \*hints,  struct addrinfo \*\*result);  成功返回0, 错误返回非零的错误码.  void freeaddrinfo(struct addrinfo \*result);  const char \*gai\_strerror(int errcode); |

地址信息结构体:

|  |
| --- |
| struct addrinfo {  int ai\_flags;  int at\_family;  int ai\_socktype;  int ai\_protocol;  char \*ai\_canonname; /\* 指向官方主机名 \*/  size\_t ai\_addrlen; /\* ai\_addr结构体的长度 \*/  struct sockaddr \*ai\_addr; /\* 指向套接字地址结构 \*/  struct addrinfo \*ai\_next;  }; |

man手册对getaddrinfo()的描述:

int getaddrinfo(const char \*node, const char \*service,

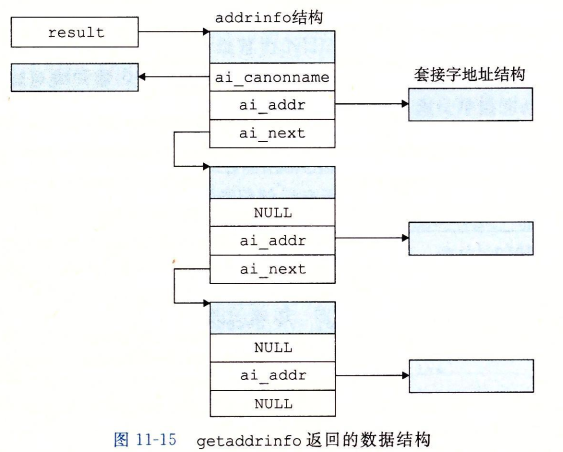
const struct addrinfo \*hints,

struct addrinfo \*\*res);

The getaddrinfo() function allocates and initializes a linked list of addrinfo structures, one for each network address that matches node and service, subject to any restrictions imposed by hints, and returns a pointer to the start of the list in res. The items in the linked list are linked by the ai\_next field.

给定host和service(套接字地址的两个组成部分), getaddrinfo返回result.

result是一个指向addrinfo结构的链表.



客户端调用getaddrinfo()后, 遍历这个链表, 尝试每个套接字地址, 直到调用socket和connnect成功.

服务端调用getaddrinfo()后, 遍历这个链表, 尝试每个套接字地址, 直到调用socket和bind成功.

freeaddrinfo()用释放该链表.

gai\_strerror()用于将错误码转换为消息字符串.

host参数可以是域名也可以是点分十进制IP地址.

service参数可以是服务名(如http)也可以是十进制端口号.

对于书本有疑惑的一段话:

如果不想把主机名转换成地址, 可以把host设置为NULL. 对services也是一样. 但是必须指令两者中至少一个.

hints参数是一个addrinfo结构体, 它能对getaddrinfo()返回的链表所包含的结构体进行限定.

hints参数只能设定以下四个字段

* ai\_family: 设置为AF\_INET只返回IPv4地址, 设置为AF\_INET6只返回IP6地址
* ai\_socktype: 设置为SOCK\_STREAM将列表限制为对每个地址最多一个addrinfo结构.
* ai\_flags字段是一个位掩码, 可以使用OR操作将各种值组合起来.

AI\_ADDRCONFIG: 不懂.

AT\_CANONNAME: getaddrinfo()将列表中第一个addrinfo的ai\_canonname指向主机官方名字.

AI\_NUMERICSERV: 强制service为端口号.

AI\_PASSIVE: 返回的套接字可能被服务器用作监听套接字. 使用该标志时host应该为NULL. 得到的套接字地址结构中的地址字段会是通配符地址, 告诉内核这个服务器会接受发送到该主机的所有IP地址请求.

getnameinfo()将一个套接字地址结构转换成相应的主机和服务名字符串. 效果与getaddrinfo()相反.

|  |
| --- |
| #include <sys/socket.h>  #include <netdb.h>  int getnameinfo(const struct sockaddr \*sa,  socklen\_t salen,  char \*host, /\* 不想要主机名时设置为NULL \*/  size\_t hostlen, /\* 不想要主机名时设置为0 \*/  char \*service, /\* 不想要服务名时设置为NULL \*/  sieze\_t servlen, /\* 不想要服务名时设置为0 \*/  int flags);  成功返回0, 错误返回非零错误代码. |

getnameinfo()将套接字地址结构sa转换成对应的主机和服务名字符串, 并将它们复制到host和service缓冲区.

flags是位掩码:

**NI\_NUMERICHOST**: 设置该标识会返回数字地址字符串.

**NI\_NUMERISERV**: getnameinfo()会默认检查/etc/services, 返回服务名. 设置该标识, 返回端口号, 而不是服务名.

getaddrinfo()和getnameinfo()的综合应用编程.

|  |
| --- |
| #include <stdio.h>  #include <sys/socket.h>  #include <netdb.h>  #include <string.h>  #define MAXLINE 1024  int main(int argc, char \*\*argv) {  struct addrinfo \*p, \*listp, hints;  char hostAddr[MAXLINE], servAddr[MAXLINE];  int rc, flags;    if(argc != 2) {  printf("Usage: %s <domain\_name>\n", argv[0]);  return -1;  }  memset(&hints, 0, sizeof(struct addrinfo));  hints.ai\_family = AF\_INET; // IPv4 address only  hints.ai\_socktype = SOCK\_STREAM; // 每个地址最多一个addrinfo结构  rc = getaddrinfo(argv[1], NULL, &hints, &listp);  if(rc != 0) {  printf("getaddrinfo error: %s\n", gai\_strerror(rc));  return -1;  }    flags = NI\_NUMERICHOST; // 返回点分十进制而不是域名  for(p = listp; p != NULL; p = p->ai\_next) {  getnameinfo(p->ai\_addr,  p->ai\_addrlen,  hostAddr,  MAXLINE,  servAddr,  MAXLINE,  flags);  printf("%s %s\n", hostAddr, servAddr); // 打印IP地址  }    freeaddrinfo(listp);    return 0;  } |

flags = NI\_NUMERICHOST;这句话好像写不写都一样.

实验结果

|  |
| --- |
| $ ./hostinfo www.baidu.com  220.181.38.150 0  220.181.38.149 0 |