




第三章

Socket编程基础

讲师：任继梅

课程目标

掌握计算机网络工作的原理

-  五层协议模型
-  掌握交换机原理
-  掌握路由器原理

掌握网络编程常用API

-  TCP编程 UDP编程

网络高级编程

-  i/o模型

课程安排



第一天

上午：计算机网络概述

下午：计算机网络原理



第二天

上午：网络原理详解

下午：TCP/IP协议栈详解



第三天

上午：Socket编程基础

下午：TCP&UDP编程基础



第四天

上午：高级网络编程

下午：高级网络编程续



第五天

上午：网络编程实战

下午：网络编程实战续

课前提问

1. 什么是网络协议？
2. 什么是IP地址？
3. 什么是端口号？
4. MAC地址又是什么东东？
5. 什么是Socket？

嵌入式

本章目标

✓ Socket套接字



✓ 套接字地址结构



✓ 值_结果参数



✓ 字节操作



✓ 地址转换



✓ 读写操作



第一节 Socket套接字

Socket套接字

定义

- socket的英文原义“孔”或“插座”，网络编程中取意“插座”，称作“套接字”
- 用于描述IP地址和端口，是一个逻辑通信链的句柄。
- 在Internet上的主机一般运行了多个服务软件，同时提供几种服务。每种服务都打开一个Socket，并绑定到一个端口上，不同的端口对应于不同的服务。
- 象一个多孔插座。一台主机犹如布满各种插座的房间，每个插座有一个编号，有的插座提供220伏交流电，有的提供110伏交流电，有的则提供有线电视节目。客户软件将插头插到不同编号的插座，就可以得到不同的服务。

Socket套接字

创建Socket套接字

```
#include <sys/types.h>           /* See NOTES */
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

- 功能：创建一个通过TCP/IP协议发送和接收网络数据的Socket
- 返回值：成功返回Socket描述符，失败 < 0
- 参数：
 - family：指明协议族（也称协议域）
 - type：指明套接字类型
 - protocol：指明协议类型，0为由family和type决定的默认协议

Socket套接字

创建socket

- AF_LOCAL在Linux中可用AF_UNIX代替

<i>family</i>	说 明
AF_INET	IPv4协议
AF_INET6	IPv6协议
AF_LOCAL	Unix域协议（见第15章）
AF_ROUTE	路由套接字（见第18章）
AF_KEY	密钥套接字（见第19章）

- 原始套接字即直接操作IP层数据

<i>type</i>	说 明
SOCK_STREAM	字节流套接字
SOCK_DGRAM	数据报套接字
SOCK_SEQPACKET	有序分组套接字
SOCK_RAW	原始套接字

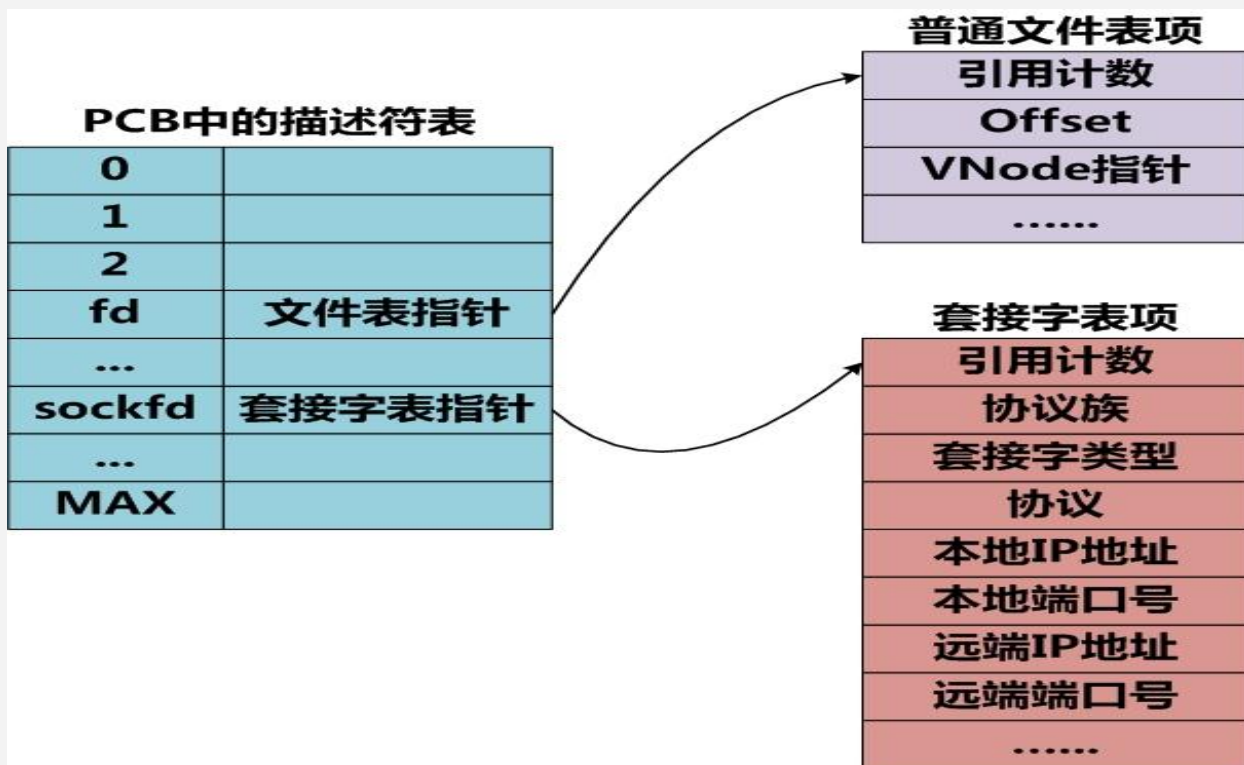
- protocol为0表示由family和type决定为默认协议

<i>protocol</i>	说 明
IPPROTO_TCP	TCP传输协议
IPPROTO_UDP	UDP传输协议
IPPROTO_SCTP	SCTP传输协议

Socket套接字

socket描述符的管理

- Linux内核将socket描述符与文件描述符进行统一管理
- 我们可以将socket看成是一种特殊的文件



Socket套接字

关闭socket

```
#include <unistd.h>

int close(int sfd);
```

- 功能：关闭套接字
- 返回值：成功返回0，失败 < 0
- 参数：
- sockfd：已打开的套接字描述符

嵌入式

地址结构

由协议族决定地址类型

- family采用AF_UNIX时采用

```
#define UNIX_PATH_MAX    108

struct sockaddr_un
{
    sa_family_t    sun_family;        /* AF_UNIX */
    char            sun_path[UNIX_PATH_MAX]; /* pathname */
};
```

- family采用AF_INET时采用

```
struct    sockaddr_in
{
    short int        sin_family;        //AF_INET
    unsigned short int    sin_port;        //端口号 网络字节序
    struct in_addr    sin_addr;        //IP地址 32位IPV4地址 网络字节序
}
```

地址结构

网际套接字地址结构

```
struct sockaddr_in
{
    short int      sin_family;    //AF_INET
    unsigned short int sin_port;  //端口号 网络字节序
    struct in_addr sin_addr;      //IP地址 32位IPV4地址 网络字节序
}
```

```
struct in_addr
{
    unsigned long int s_addr; //IP地址 32位IPV4地址 网络字节序
}
```

地址结构

通用套接字地址结构

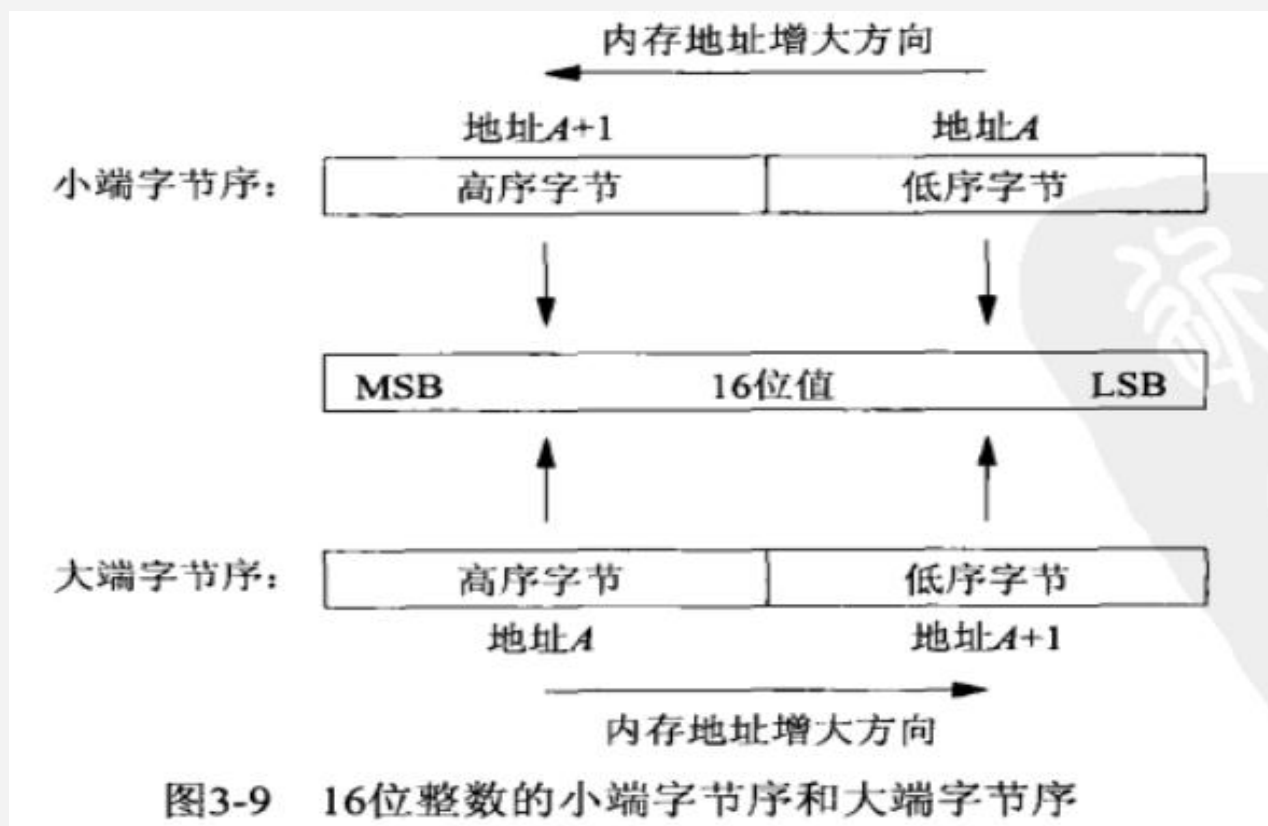
- 后续socket某些操作函数为了兼容多种地址类型往往采用通用地址结构

```
struct sockaddr
{
    short int    sin_family; //协议族
    char         sa_data[14]; //协议指定地址
}
```

字节操作



字节序



字节操作

测试字节序

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    unsigned short n = 0x1234;
    unsigned char *m = (unsigned char *)&n;

    if(*m == 0x34)
    {
        printf("This Host is Little-Endian\n");
    }
    else
    {
        printf("This Host is Big-Endian\n");
    }
    return 0;
}
```


字节操作

主机字节序&网络字节序

- 主机字节序：某个给定的主机系统所用的字节序，它由CPU决定.
- 网络字节序：网络协议指定的字节序，网际协议使用大端字节序来传送多字节整数.

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);
```

- h表示host，n表示network，s表示short，l表示long

字节操作

常用字节操作函数

```
#include <string.h>

void *memset(void *s, int c, size_t n);

void *memcpy(void *dest, const void *src, size_t n);

int memcmp(const void *s1, const void *s2, size_t n);

#include <strings.h>

void bzero(void *s, size_t n); /*等价于memset(dest, 0, nbytes)*/
```

地址转换

十进制与32位地址间的相互转换

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int inet_aton(const char *cp, struct in_addr *inp);

char *inet_ntoa(struct in_addr in);
```

- 返回值：字符串有效返回1，否则返回0
- 注：
 - struct in_addr inaddr均指网络字节序的32位IP地址
 - inet_ntoa函数的结果位于静态数据区

地址转换

域名与地址转换

```
#include <netdb.h>
extern int h_errno;

struct hostent *gethostbyname(const char *name);

#include <sys/socket.h>          /* for AF_INET */

struct hostent *gethostbyaddr(void *addr, socklen_t len, int type);
```

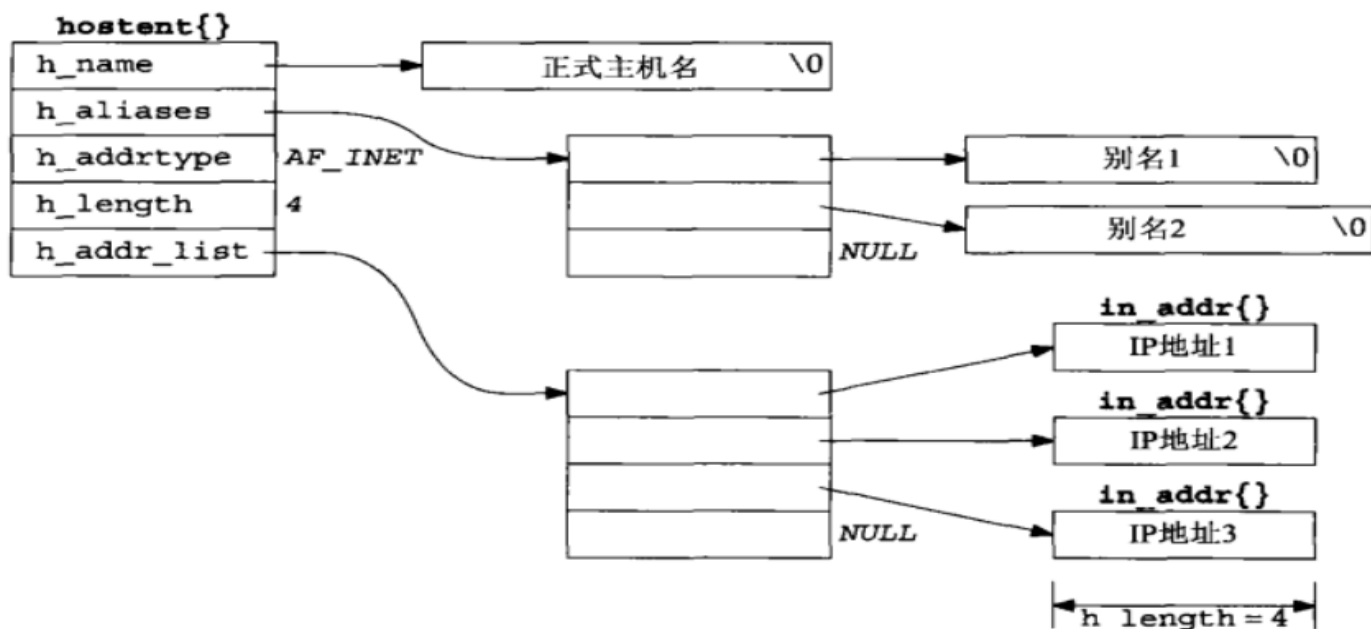
- ☰ 注：addr不是点分字符串而是struct in_addr *
- ☰ 注：
 1. struct in_addr inaddr均指网络字节序的32位IP地址
 2. inet_ntoa函数的结果位于静态数据区

地址转换



域名与地址转换

```
struct hostent
{
    char    *h_name;           /* official name of host */
    char    **h_aliases;      /* alias list */
    int      h_addrtype;       /* host address type */
    int      h_length;         /* length of address */
    char    **h_addr_list;    /* list of addresses */
}
```



地址转换

域名与地址转换例子：

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    struct hostent * host = NULL;
    struct in_addr *pinaddr = NULL;
    struct in_addr **ppinaddr = NULL;
    char *ip = NULL;
    char **ppaliases = NULL;

    host = gethostbyname("www.sina.com.cn");
    if(host == NULL)
    {
        perror("gethostbyname()");
        return -1;
    }
}
```

地址转换

📄 域名与地址转换例子：

```
printf("h_name=%s\n", host->h_name);
ppaliases = host->h_aliases;
while(*ppaliases != NULL)
{
    printf("aliases = %s\n", *ppaliases);
    ppaliases++;
}

ppinaddr = (struct in_addr **)host->h_addr_list;
while(*ppinaddr != NULL)
{
    pinaddr = *ppinaddr;
    ip = inet_ntoa(*pinaddr);
    printf("ip=%s\n", ip);

    ppinaddr++;
}

return 0;
}
```

本节完
谢谢

上 嵌

课程总结



本节课程内容

- Socket套接字
- 套接字的地址结构
- 值_结果参数
- 字节操作
- 地址转换



下节课程

- TCP UDP编程
- TCP客户端和服务气端
- UDP客户端和服务服务器端

嵌入式