第七章：套接字SOCKET

目标:

本章旨在向学员介绍Linux系统下套接字的使用及方法：

1）掌握面向连接的套接字编程方法

2）掌握非连接的套接字编程方法

时间：3 学时

教学方法：讲授PPT、实例练习

7.1 关于SOCKET

功能

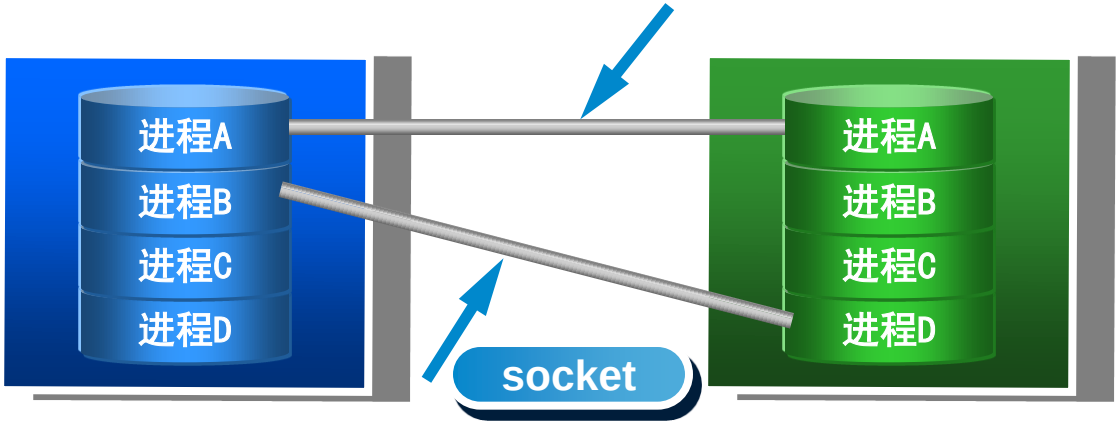
为不同机器上的两个进程之间提供通信机制

实现

在硬件层通过网络设备连接，在软件层通过标准的网络协议集TCP或UDP

7.1 关于SOCKET

- socket提供了不同主机进程间数据通信的机制



7.2 套接字

- IP地址:将数字点形式表示的字符串IP地址与32位网络字节顺序的二进制形式的IP地址进行转换.

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

in_addr_t inet_addr(const char *ip_address);
```

返回值in_addr_t调用成功后，将返回IP地址，错误返回-1

例子：

```
in_addr_t server;
server = inet_addr(“192.168.0.1”);
```

7.2 套接字

- 端口

网络套接字结构定义如下：

```
#include <netinet/in.h>
struct sockaddr_in
{
  sa_family_t  sin_family;     /* internet address family */
  in_port_t      sin_port;        /*port number */
  struct  in_addr  sin_addr;   /* holds the IP address */
  unsigned char  sin_zero[8]  /*filling */
 };
```

- sin_family指代协议族，在socket编程中只能是AF_INET
- sin_port存储端口号（使用网络字节顺序）
- sin_addr存储IP地址，使用in_addr这个数据结构
- sin_zero是为了让sockaddr与sockaddr_in两个数据结构保持大小相同而保留的空字节。

7.2 套接字

- 主机字节序与网络字节序
- 计算机数据存储有两种字节优先顺序：高位字节优先和低位字节优先。Internet上数据以高位字节优先顺序在网络上传输，所以对于在内部是以低位字节优先方式存储数据的机器，在Internet上传输数据时就需要进行转换，否则就会出现数据不一致。 这些函数将16位和32位整数在主机字节序和网络字节序之间进行转换

```
#include <netinet/in.h>

unsigned long int htonl(unsigned long int hostlong);
unsigned short int htons(unsigned short int hostshort);
unsigned long int ntohl(unsigned long int netlong);
unsigned short int ntohs(unsigned short int netshort);
```

·htonl()：把32位值从主机字节序转换成网络字节序
·htons()：把16位值从主机字节序转换成网络字节序
·ntohl()：把32位值从网络字节序转换成主机字节序
·ntohs()：把16位值从网络字节序转换成主机字节序

```
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
server_address.sin_port = htons(9734);
```

7.2 套接字

- 地址转换打印函数inet_ntoa

将网络字节转换为可打印四点表示法格式的IP地址字符串

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

char *inet_ntoa(struct in_addr in)
```

- gethostname函数

获得当前主机的名字，存入name参数中

namelength：缓冲区的长度。

```
#include <unistd.h>

int gethostname(char *name, int namelength);
```

7.2 套接字

- socket函数：功能为创建通信连接句柄

```
#include <sys/socket.h>

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

参数domain：指定套接字的类型

AF_INET 网络上的套接字

AF_UNIX 进程都运行于同一台机器时

参数type：指定了建立的套接字是用于连接模型还是无连接模型

SOCK_STREAM 连接模型，默认TCP协议

SOCK_DGRAM 无连接模型，默认UDP协议

参数protocol：指定所使用的协议

该值一般被设为0，表示默认协议

7.2 套接字

- bind(绑定）

把电脑上真正的网络地址与一个套接字标识符关联起来

```
#include <sys/types.h>
#include <sys/socket.h>

int bind (int sockfd, const struct sockaddr *address, size_t add_len);
```

参数sockfd:是从socket系统调用返回的文件描述符
参数address:指向套接字结构的指针。
参数add_len:存储套接字实际使用的地址指针的大小

bind调用成功返回0，错误返回-1

bind函数将socket与本机上的一个端口相关联，随后就可以在该端口监听服务请求。

7.2 套接字

- listen(监听)

绑定之后，在任何客户端系统可以连接到新建立的服务器端点之前，服务器必须设定为等待连接。

```
#include <sys/socket.h>

int listen(int sockfd, int queue_size);
```

参数sockfd：sock系统调用返回的文件描述符
参数queue_size：允许多少个连接请求排入队列

7.2 套接字

- accept (接受连接)

当服务器收到客户端connect请求时，必须建立一个全新的套接字来处理这个特定的通信。第一个套接字只用来建立通信，第二个套接字由accept完成。

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *address, size_t *add_len);
```

参数sockfd：socket系统调用返回的文件描述符
参数address：将客户端信息填充。
参数vadd_len：保存实际复制的字节数。

7.2 套接字

- connect(请求连接)

客户程序通过一个未命名套接字和服务器监听套接字之间建立连接的方法来连接到服务器。通过connect调用来完成

```
#include <sys/socket.h>

int connect(int socket, const struct sockaddr *address, size_t address_len);
```

参数socket：指定的套接字将连接到参数address指定服务器套接字
参数address_len：指向结构长度由参数address_len指定

7.2 套接字

- 接收数据

recv调用从指定的文件描述符读取数据，存放到buffer中，buffer的长度是length。

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t recv(int sockfd, void *buffer, size_t length,int flags);
```

flags参数：影响数据被接收的方式：

MSG_PEEK 进程查看数据但并不接收。

MSG_OOB 普通数据被忽略，进程接收“带外数据”，例如中断信号

MSG_WAITALL recv调用只有接收到足够长度的数据时才返回

7.2 套接字

- 发送数据

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t send(int sockfd, const void *buffer, size_t length, int flags);
```

flags参数影响数据发送的方式：

MSG_OOB　发送“带外数据”

7.2 套接字

- 面向非连接的发送与接受函数

```
ssize_t recvfrom(int sockfd, void *message, size_t length,int flags,
struct sockaddr *send_addr size_t *add_len);

ssize_t sendto(int sockfd, const void *message, size_t length int
flags,const struct sockaddr *dest_addr,size_t dest_len);
```

参数send_addr设置为NULL recvfrom调用与recv调用工作方式相同

参数message为接受或发送的数据，length为读写缓冲区的长度

参数flags与recv调用的参数相同

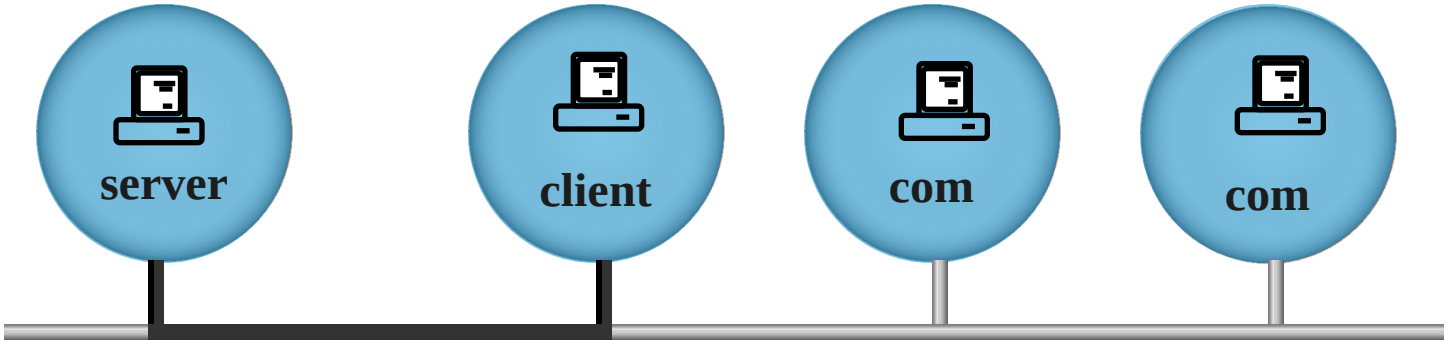
最后的参数为发送或接受的地址

7.2 套接字

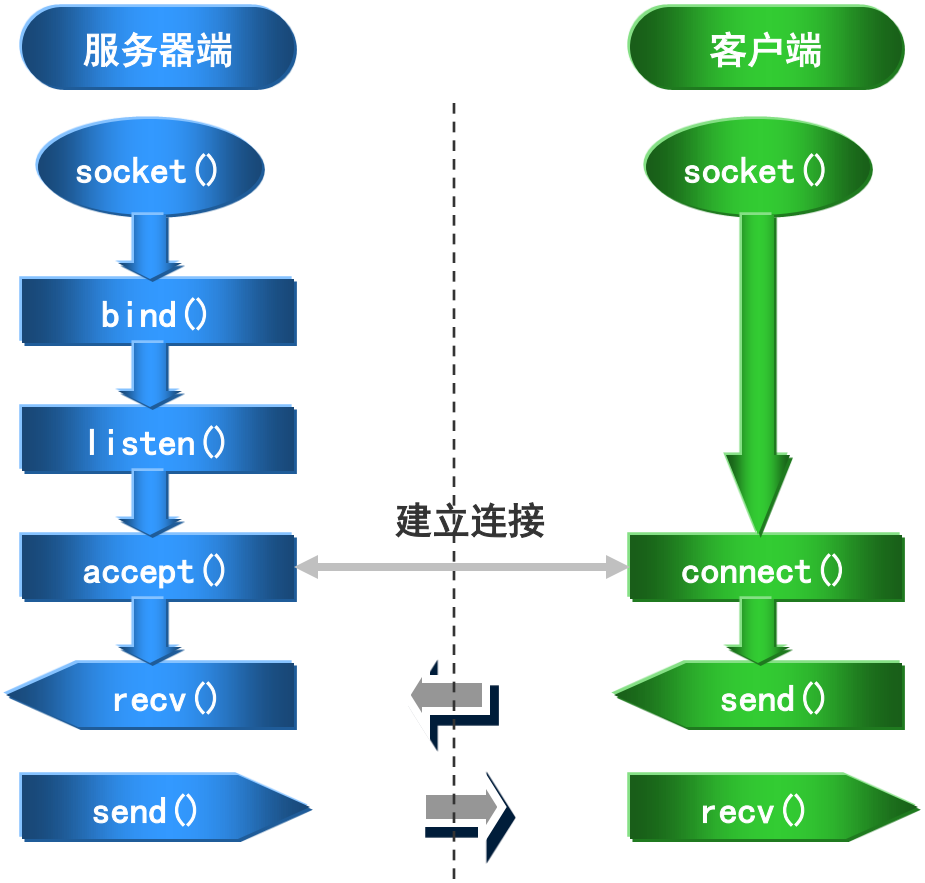
- close函数：关闭套接字

  连接类型为SOCK_STREAM 内核将保证数据被发往接受进程
  连接类型为SOCK_DGRAM 套接字将会被马上关闭

7.3 面向连接套接字



7.3 面向连接套接字


服务器端
客户端
socket()
socket()
bind()
listen()
建立连接
accept()
connect()
recv()
send()
send()
recv()

7.3 面向连接套接字

- 面向连接的例程代码

udprecv.c 头文件部分

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa.inet.h>
#define myport 319
char buf[200];
```

7.3 面向连接套接字

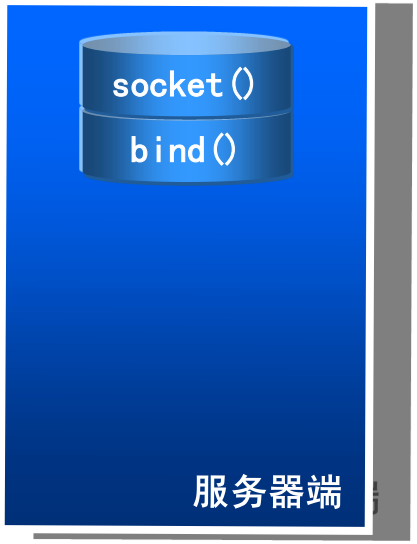
- 创建socket代码部分

```
int main(int argc , char *argv[])
 {
    int sockfd,new_sockfd;
    struct sockaddr_in sin_addr,pin_addr;
    int len, pin_addr_size,i;
    if ((sockfd = socket(AF_INET, SOCK_STREAM,
0)) <0)
    {
      printf("can't create socket\n");
      exit(1);
     }
```

7.3 面向连接套接字

- bind绑定端口代码部分

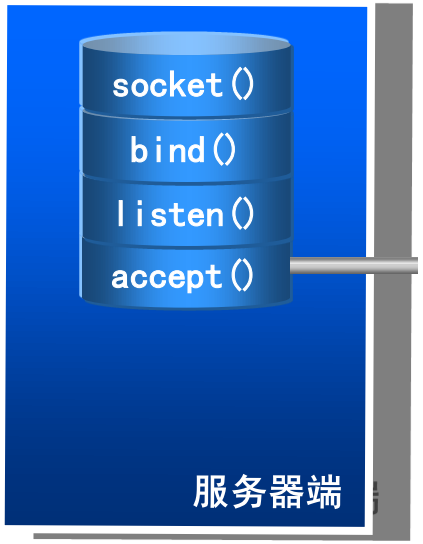
```
memset( &sin_addr, 0, sizeof(sin_addr));
sin_addr.sin_family = AF_INET;
sin_addr.sin_addr.s_addr = htonl(INADDR_ANY);
sin_addr.sin_port = htons(myport);
if (bind(sockfd, (struct sockaddr *)
                &sin_addr, sizeof(sin_addr))<0)
{
  printf("can't bind socket\n");
  exit(1);
}
```



7.3 面向连接套接字

- listen 及 accept 部分代码

```
if (listen(sockfd, 5) < 0 )
{
  printf("listen error\n");
  exit(1);
 }
While (1)
{
  if ((new_sockfd = accept (sockfd, (struct sockaddr *)
                          &pin_addr,&pin_addr_size)) < 0)
  {
      printf("accept eroor\n");
      exit(1);
  }
```



7.3 面向连接套接字

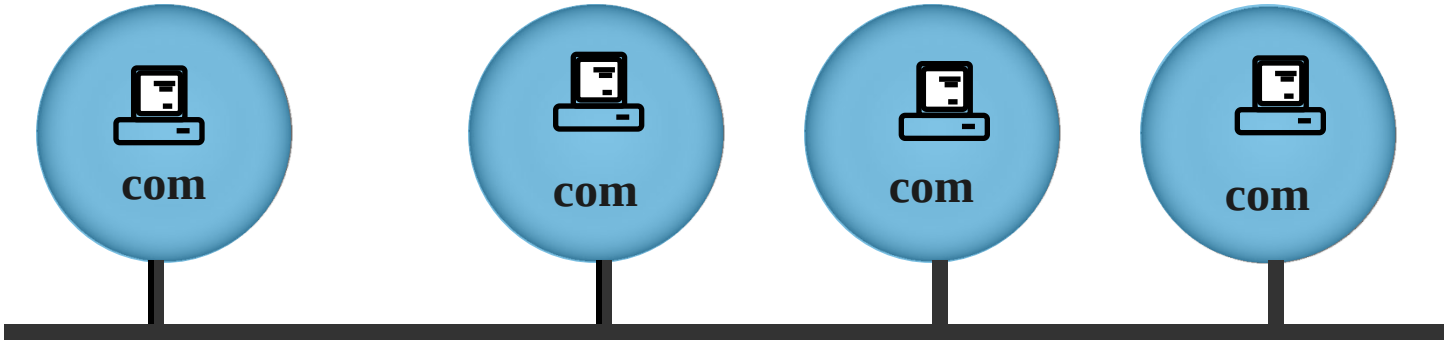
- recv读取接受数据代码部分

```
if(recv(new_sockfd, buf, 200, 0) == -1)
{
   printf("can't receive packet\n");
   exit(1);
}
 printf("received from client:%s\n", buf);
 close(new_sockfd);
}
return 0;
}
```

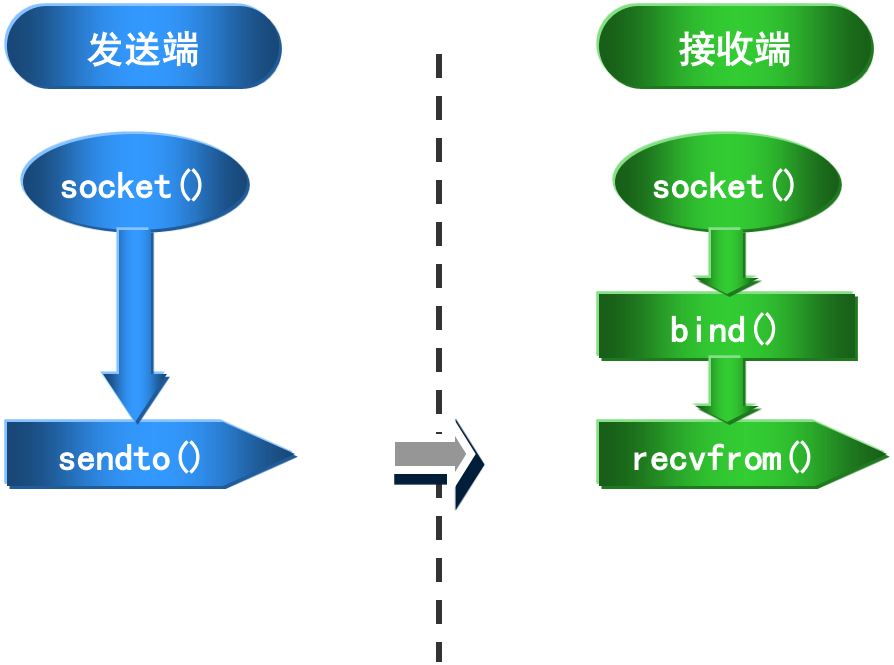
7.3 面向连接套接字

- 练习：编写客户端程序，尝试与服务器端进行通信。

7.4 面向非连接套接字



7.4 面向非连接套接字



7.4 面向非连接套接字

- 接收端udprecv.c头文件代码部分

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int myport = 320;
```

7.4 面向非连接套接字

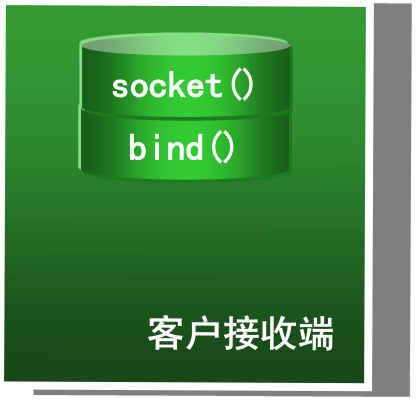
- 创建socket代码部分

```
int main()
{
  int sin_len;
  char buf[200];
  int socketfd;
  struct sockaddr_in sin_addr;
  bzero(&sin_addr, sizeof(sin_addr));
  sin_addr.sin_family = AF_INET;
  sin_sin_addr.s_addr = htonl(INADDR_ANY);
  sin.sin_port = htons(myport);
  sin_len = sizeof(sin);
  if (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == 1)
  {
     printf("Error opening socket\n");
     exit(1);
   }
```

7.4 面向非连接套接字

- bind绑定端口代码部分

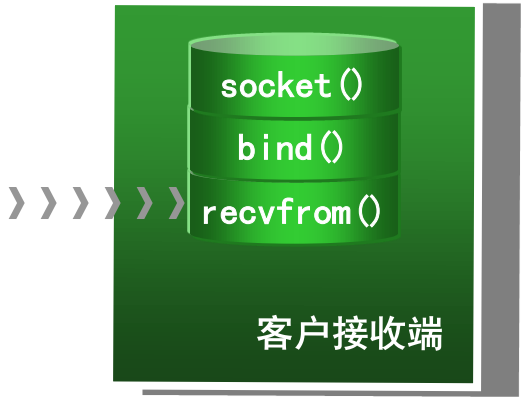
```
if (bind(sockfd,(struct sockaddr *)&sin_addr,
                        sizeof(sin_addr)) == -1)
{
   printf("Error in bind\n");
   exit(1);
}
```



7.4 面向非连接套接字

- recvfrom 接受数据代码部分

```
while(1)
{
  if (recvfrom(sockfd, buf , sizeof(buf),
   0,(struct sockaddr*)&sin_addr, &sin_len) == -1)
  {
    printf("receive message from server:%s\n", buf);
   }
}
//never run to here
close(sockfd);
return 0;
}
```



7.4 面向非连接套接字

- 练习：编写发送端程序，尝试与接收端程序进行通信。

7.5 两种模型的区别

相同	服务器进程都需要创建套接字，并把自己的本地地址绑定到这个套接字上
不同	连接模型中，服务器接下来必须监听进入的连接。 无连接模型中，就不需要这一步，客户端会做更多的工作
特点	连接模型中客户端只需要连接到服务器就可以了，无连接模型中，客户端必须创建套接字并绑定地址

7.6 时间同步程序

- 练习：设计并编写时钟同步程序，使得客户端电脑可以与服务端电脑时间进行同步，只需要在客户端电脑输出服务器端电脑时间即可，不需要进行实际的同步操作。

- 时间相关的系统函数使用：
- long now;　　定义保存时间值的变量
- time(&now);　　获得当前系统时间数值
- ctime(&now);　　函数返回值为指定格式的字符类型时间数据