Socket 编程

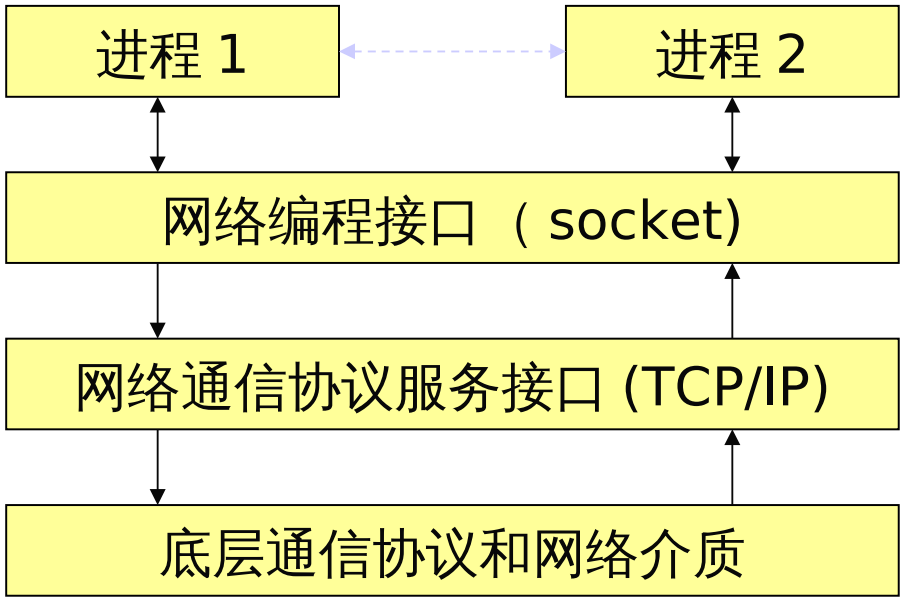
2012-09

课程内容

- 套接字概述
- 流套接字编程模型
- 数据包套接字编程模型
- 附录：常用套接字函数使用说明

套接字基础

- 套接字是一种网络 API， 程序员可以用之开发网络程序。



套接字类型

- Linux 支持多种套接字类型，即应用程序希望的通信服务类型
  - SOCKET_STREAM ：双向可靠数据流，对应 TCP
  - SOCKET_DGRAM ：双向不可靠数据报，对应 UDP
  - SOCKET_RAW ：是低于传输层的低级协议或物理网络提供的套接字类型，可以访问内部网络接口。例如接收和发送 ICMP 报

套接字地址结构（IPv4)

- 大多数套接字函数需要一个指向套接字地址结构的参数，每个协议族都定义它自己的套接字地址结构，一般以”sockaddr_”开头，并以协议簇为后缀。（netinet/in.h）

```
typedef  uint32_t  in_addr_t;
typedef  uint16_t  in_port_t;
typedef  unsigned short  sa_family_t;

struct  in_addr{
   in_addr_t  s_addr;
};
```

```
struct  sockaddr_in {
   uint8_t          sin_len;
   sa_family_t      sin_family;
   in_port_t        sin_port;
   struct  in_addr  sin_addr;
   char             sin_zero[8];
};
```

这些结构在不同的系统上都有所不同

套接字地址结构（IPv6)

- IPv6 地址为 128 位。（netinet/in.h）

```
typedef  uint16_t  in_port_t;
typedef  unsigned short  sa_family_t;

struct  in6_addr{
   uint8_t          s6_addr[16];
};
```

```
struct  sockaddr_in6{
   uint8_t          sin6_len;
   sa_family_t      sin6_family;
   in_port_t        sin6_port;
   uint32_t         sin6_flowinfo;
  struct  in6_addr  sin6_addr;
};
```

sin6_flowinfo 成员分成三个字段：

- 低 24 位是流量标号；
- 下 4 位是优先级；
- 再下 4 位保留

通用套接字地址结构

- 由于套接字函数需接收来自不同协议的地址结构， ANSI 的办法是使用通用的指针类型，即（ void * ） . 套接字函数方法是定义一个通用的套接字地址结构。 <sys/socket.h>

```
struct  sockaddr{
   uint8_t          sa_len;
   sa_family_t      sa_family;
   char             sa_data[14];
};
```

这就要求调用套接字函数时，需将指向特定于协议的地址结构的指针类型转换成指向通用的地址结构的指针，

如： **struct sockaddr_in serv**

bind(sockfd, (struct sockaddr *)&serv, sizeof(serv));

网络字节顺序与主机字节顺序

- Big-endian 与 little endian

htons(u_short hostshort)
htonl(u_long hostlong)
ntohs(u_short netshort)
ntohl(u_long netlong)

TCP 套接字编程（ cont.）

TCP 套接字编程

- 实现 TCP 套接字基本步骤分为服务器端和客户端两部分：
- 服务器端

a. 创建套接字；
b. 绑定套接字；
c. 设置套接字为监听模式，进入被动接受连接状态；
d. 接受请求，建立连接
e. 读写数据
f. 终止连接

TCP 服务器模板

```
int main(void)
{
    int sockfd,connect_sock;
    if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)  {

        perror("create socket failed.");
        exit(-1);
    }
    /* bind sockfd to some address */
    /* listen */
    ……
    loop  {

      if((connect_sock=accept(sockfd,NULL,NULL))==-1)  {
                    perror("Accept error.");
            exit(-1);
          }
       /* read and process request */
        close(connect_sock);
      }
    close(sockfd);
}
```

TCP 套接字编程（ cont.）

- 客户端步骤
  a. 创建套接字
  b. 与远程服务器建立连接
  c. 读 / 写数据；
  d. 终止连接

TCP 客户模板

```
/* include some header files */
int main(void)
{
    int sockfd;
    if((sockfd=socket(AF_INET,SOCK_STREAM,0))=-1)
    {
        perror("Create socket failed.");
        exit(-1);
    }
    /* connect to server */
    ……
    /* send requst and receive response */
    ……
    close(sockfd);
}
```

无连接的服务

UDP 编程的适用范围

- 部分满足以下几点要求时，应该用 UDP
  - 面向数据报
  - 网络数据大多为短消息
  - 拥有大量 Client
  - 对数据安全性无特殊要求
  - 网络负担非常重，但对响应速度要求高
- 例子： ICQ 、视频点播

具体编程时的区别

- socket() 的参数不同
- UDP Server 不需要调用 listen 和 accept
- UDP 收发数据用 sendto/recvfrom 函数
- TCP ：地址信息在 connect/accept 时确定
  UDP ：在 sendto/recvfrom 函数中每次均需指定地址信息

UDP 套接字编程

- 实现 TCP 套接字基本步骤分为服务器端和客户端两部分：
- 服务器端
  a. 创建套接字；
  b. 绑定套接字；
  c. 读取客户端发过来的数据；
  d. 向客户端写数据
  e. 终止连接

UDP 服务器模板

```
int main(void)
{
    int sockfd,connect_sock;
    if((sockfd=socket(AF_INET,SOCK_DGRAM,0))==-1)   {

            perror(“create socket failed.”);
            exit(-1);
    }
    /* bind sockfd to some address */
    /* read */
    recvform();
    sendto();
                }
    close(sockfd);
}
```

UDP 套接字编程（ cont.）

- 客户端步骤

  a. 创建套接字
  b. 绑定本地 IP 地址与端口
  c. 向服务器发送数据；
  d. 读取服务器发来的数据
  e. 终止连接

UDP 客户模板

```
/* include some header files */
int main(void)
{
    int sockfd;
    if((sockfd=socket(AF_INET,SOCK_DGRAM,0))=-1)
    {
        perror("Create socket failed.");
        exit(-1);
    }
   if (-1==bind(sockfd,(struct
    sockaddr*)&client,sizeof(client)))
        ……
    /* send requst and receive response */
    ……
    close(sockfd);
}
```

本次试验内容

- 结合 big-endian 和 little-endian 的概念编写一个程序，判断自己试验用机的字节存储结构；
- 借鉴老师提供的采用 TCP 协议实现的通讯程序，采用 UDP 协议实现相同功能的程序；

基本套接字函数 – socket

#include <sys/socket.h>

int socket(int family, int type, int protocol)

返回：非负套接字（ sockfd ）– 成功； -1 – 出错。

family: 通信域； type: 套接字类型 ; protocol ：一般为 0 。

family		type	
AF_INET	IPv4 协议	SOCK_STREAM	字节流套接口
AF_INET6	IPv6 协议	SOCK_DGRAM	数据报套接口
AF_UNIX	unix 域协议	SOCK_RAW	原始套接口

Protocol ：指明此 socket 请求所使用的协议，一般采用默认的 0， 表示按给定的域和套接字类型选择默认协议。也可以使用如下相关符号常数来表示：
IPPROTO_TCP: 表示 TCP 协议
IPPROTO_UDP: 表示 UDP 协议

基本套接字函数 – bind

#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr, socklen_len len)

返回： 0 – 成功； -1 – 出错并置 errno

- 该函数指明套接字将使用本地的哪一个协议端口进行数据传送（ IP 地址和端口号），注意：协议地址 addr 是通用地址。
- Len 是该地址结构（第二个参数）的长度。
- 一般而言，服务器调用此函数，而客户则很少调用它。

bind 函数的用法

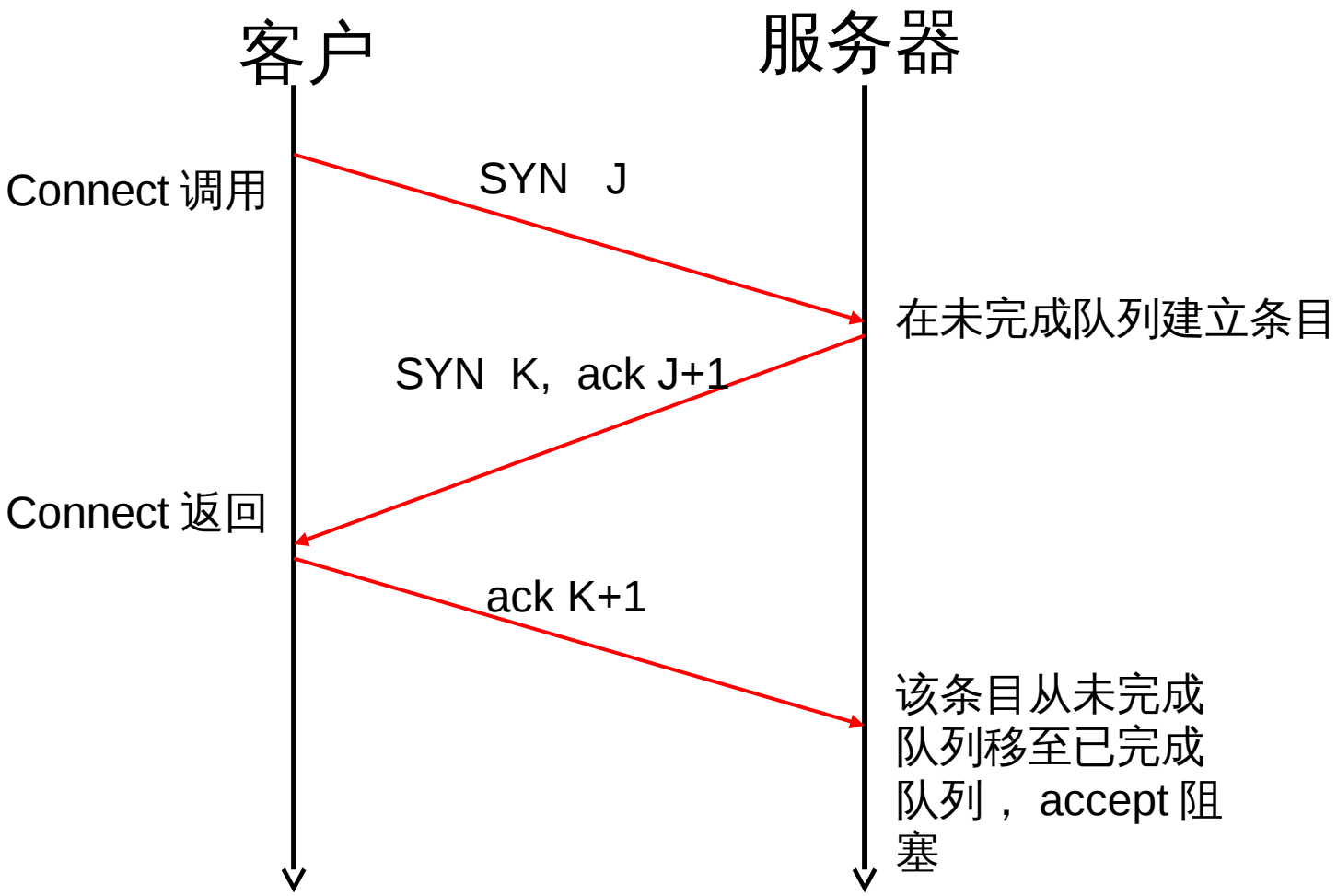
```
  ...
  struct sockaddr_in           addr;
  int port = 1234;
  int opt = SO_REUSEADDR;
  setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
bzero(&server,sizeof(server));
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = htonl(INADDR_ANY);
  addr.sin_port = htons(port);

  if (bind(fd, (struct sockaddr *)&addr, sizeof(addr)) == -1)
  {
        /*  错误处理   */
  }
```

基本套接字函数 – listen

```
#include <sys/socket.h>
int listen(int sockfd, int backlog)
```

返回： 0 –成功； -1 –出错并置 errno 值；

- 函数 listen 仅被服务器调用 , 它完成两件事情：
  - 函数 listen 将未连接的套接字转化成被动套接字，指示内核应接受指向此套接字的连接请求；
  - 函数的第二个参数规定了内核为此套接字排队的最大连接个数，但与系统所允许的最大连接数没有关系。
- 对于给定的监听套接字，内核要维护两个队列
  - 未完成连接队列
  - 已完成连接队列，但还没有被应用程序所接受
  - 两个队列之和不超过 backlog;



TCP 三路握手和监听套接口的两个队列

基本套接字函数 - connect

```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

返回： 0 - 成功； -1 - 出错；

- 函数 connect 激发 TCP 的三次握手过程；仅在成功或出错返回；错误有以下几种情况：
  - 如果客户没有收到 SYN 分节的响应（总共 75 秒，这之间需要可能需要重发若干次 SYN ），则返回 ETIMEDOUT 。
  - 如果对客户的 SYN 的响应是 RST ，则表明该服务器主机在指定的端口上没有进程在等待与之相连。函数返回错误 ECONNREFUSED ；
  - 如果客户发出的 SYN 在中间路由器上引发一个目的地不可达 ICMP 错误，客户上的内核保存此消息，并按第一种情况，连续发送 SYN ，直到规定时间，返回保存的消息（即 ICMP 错误）作为 EHOSTUNREACH 或 ENETUNREACH 错误返回给进程。

基本套接字函数 – accept

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```

返回：非负描述字（ connfd ） – OK ； -1 – 出错；

- accept 函数由 TCP 服务器调用；从已完成连接队列头返回下一个已完成连接；如果该队列空，则进程进入睡眠状态。
- 函数返回的套接字为已连接套接字，应与监听套接字区分开来
- 该函数最多返回三个值：一个既可能是新套接字也可能是错误指示的整数，一个客户进程的协议地址（由 cliaddr 所指），以及该地址的大小；也就是说，服务器可以通过参数 cliaddr 来得到请求连接并获得成功的客户的地址和端口号；

基本套接字函数 – close

```
#include <unistd.h>
int close(int sockfd);
```

返回 :0 – OK ； -1 – 出错；

- close 函数缺省功能是将套接字做上“已关闭”标记，并立即返回到进程。这个套接字不能再为该进程所用。
- 正常情况下， close 将引发向 TCP 的四分节终止序列，但在终止前将发送已排队的数据；
- 如果套接字描述符访问计数在调用 close 后大于 0 （在多个进程共享同一个套接字的情况下），则不会引发 TCP 终止序列（即不会发送 FIN 分节）；

基本套接字函数 - read

#include <unistd.h>

int read(int fd, char *buf, int len);

返回 : 大于 0 - 读写字节大小； -1 - 出错；

- 调用函数 read 时，有如下几种情况：
  - 套接字接收缓冲区接收数据，返回接收到的字节数；
  - tcp 协议收到 FIN 数据，返回 0 ；
  - tcp 协议收到 RST 数据，返回 - 1 ，同时 errno 为 ECONNRESET ；
  - 进程阻塞过程中接收到信号，返回 - 1 ，同时 errno 为 EINTR 。

read （ connfd ， buff ， bufflen ）；

基本套接字函数 - write

```
#include <unistd.h>
int write(int fd, char *buf, int len);
```

返回 : 大于 0 - 写字节大小； -1 - 出错；

- 调用函数 write， 有如下几种情况：
  - 套接字发送缓冲区有足够空间，返回发送的字节数；
  - tcp 协议接收到 RST 数据，返回 - 1， 同时 errno 为 ECONNRESET ； ；
  - 进程阻塞过程中接收到信号，返回 - 1， 同时 errno 为 EINTR 。

  write （ connfd， buff， strlen （ buff ））；

数据传输函数 – send

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t send (int fd, const void *msg, size_t len, int flags);
```

返回：非 0 – 发送成功的数据长度； -1 – 出错；

- flags 是传输控制标志，其值定义如下：
  - 0 ：常规操作，如同 write() 函数
  - MSG_OOB ， 发送带外数据 (TCP 紧急数据) 。
  - MSG_DONTROUTE ：忽略底层协议的路由设置，只能将数据发送给与发送机处在同一个网络中的机器上。

数据传输函数 – recv

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recv(int fd, void *buf ,size_t len, int flags);
```

返回：大于 0 表示成功接收的数据长度； 0: 对方已关闭， -1: 出错。

- flags 是传输控制标志，其值定义如下：
  - 0 ：常规操作，如同 read() 函数；
  - MSG_PEEK ：只查看数据而不读出数据，后续读操作仍然能读出所查看的该数据；
  - MSG_OOB ：忽略常规数据，而只读带外数据；
  - MSG_WAITALL ：等待直到所有的数据可用，仅 SOCK_STREAM 。此时 len 表示的是等待数据的长度，而不再是 buf 的长度。

实验题目

- 在老师给出的事例程序的基础上 ， 采用用数据报的方式改写客户端与服务器端程序，实现一个与实例程序相同的网络通讯程序。