第6章
网络程序设计

Socket概述

Socket

- **网络协议的作用**
- **协议栈实现：传输层和传输层以下协议在内核中实现**
- **UNIX提供给应用程序使用网络功能的方法**
  - 将设备和通信管道组织成文件方式，创建方式不同，访问方法相同
    - 终端设备
    - 管道
    - 通信服务Socket
  - TLI编程接口
- **Socket编程接口面向网络通信，不仅仅用于TCP/IP**
  - 利用虚拟loopback接口(127.0.0.1)，可实现同台计算机进程间通信

TCP与UDP

- **TCP**
  - 面向连接
  - 可靠
  - 字节流传输
    - 不保证报文边界
- **UDP**
  - 面向数据报
  - 不可靠
    - 错报，丢报，重报，乱序，流量控制
  - 数据报传输
  - 广播和组播

网络字节顺序

- **CPU字节顺序**
  - Big Endian (大尾)
    - Power PC，SPARC，Motorola
  - Little Endian (小尾)
    - Intel X86
- **网络字节顺序**
  - 与X86相反
- **网络字节转换的库函数**
  - htonl ntohl 四字节整数(long)
  - htons ntohs 两字节整数(short)

TCP客户-服务器程序

客户端程序：client.c(1)

```
#define SIZE 8192
#define PORT_NO 12345
int main(int argc, char *argv[])
{
    int sock, len;
    struct sockaddr_in name;
    unsigned char sbuf[SIZE];
    if (argc < 2)  ...
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) ...
    name.sin_family = AF_INET;
    name.sin_addr.s_addr = htonl(inet_network(argv[1]));
    name.sin_port = htons(PORT_NO);
    if (connect(sock, &name, sizeof(name)) < 0) {
        perror("\nconnecting server stream socket");
        exit(1);
    }
    printf("Connected.\n");
```

客户端程序：client.c(2)

```
  for(;;) {
        if (fgets(sbuf, SIZE, stdin) == NULL) break;
        if (write(sock, sbuf, strlen(sbuf)) < 0) {
            perror("sending stream message");
            exit(1);
        }
    }
    close(sock);
    printf("Connection closed.\n\n");
    exit(0);
}
```

客户端程序

- **创建文件描述符socket**
- **建立连接connect**
  - 进程阻塞，等待三次握手成功
- **端点名的概念：IP地址+端口号**
  - 本地端点名
  - 远端端点名
- **发送数据**
  - 发送速率大于通信速率，进程会被阻塞
- **关闭连接**

服务端程序：server0.c(1)

```
#define PORT_NO  12345
 int main(void)
 {
    int admin_sock, data_sock, nbyte, i;
    struct sockaddr_in name;
    char buf[8192];
    admin_sock = socket(AF_INET, SOCK_STREAM, 0);
    name.sin_family = AF_INET;
    name.sin_addr.s_addr = INADDR_ANY;
    name.sin_port = htons(PORT_NO);
    bind(admin_sock, &name, sizeof(name));
    listen(admin_sock, 5);

    data_sock = accept(admin_sock, 0, 0);
    printf("Accept connection\n");
```

服务端程序：server0.c(2)

```
for (;;) {
        nbyte = read(data_sock, buf, sizeof(buf));
        if (nbyte == 0) {
            printf("*** Disconnected.\n");
            close(data_sock);
            exit(0);
        }
        for (i = 0; i < nbyte; i++)
            printf("%c", buf[i]);
    }
}
```

服务端程序

- **创建文件描述符socket**
- **bind**
  - 设定本地端点名
  - 也可以用在客户端程序
- **Listen**
  - 进程不会在此被阻塞，仅仅给内核一个通知
- **accept**
  - 进程会在这里阻塞等待新连接到来
- **创建新进程时的文件描述符处理**
- **问题：不能同时接纳多个连接**
  - 解决方法
    - 多进程并发处理
    - 单进程并发处理

多进程并发：server1.c(1)

```
#define PORT_NO  12345
int main(void)
{
    int admin_sock, data_sock, pid, name_len;
    struct sockaddr_in name, peer;
    admin_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (admin_sock < 0) {
        perror("create stream socket");
        exit(1);
    }
    name.sin_family = AF_INET;
    name.sin_addr.s_addr = INADDR_ANY;
    name.sin_port = htons(PORT_NO);
    if (bind(admin_sock, &name, sizeof(name)) < 0) {
        perror("binding stream socket");
        exit(1);
    }
    listen(admin_sock, 5);
    signal(SIGCLD, SIG_IGN);
```

多进程并发：server1.c(2)

```
for (;;) {
        name_len = sizeof(peer);
        data_sock = accept(admin_sock, &peer, &name_len);
        if (data_sock < 0) continue;
        printf("Accept connection from %s:%d\n",
              inet_ntoa(peer.sin_addr), ntohs(peer.sin_port));
        pid = fork();
        if (pid > 0) {  /* parrent process */
            close(data_sock); /* 不可省略，原因有2 */
        } else if (pid == 0) { /* child process   */
            char fd_str[16];
            close(admin_sock);
            sprintf(fd_str, "%d", data_sock);
            execlp("./server1a", "./server1a", fd_str, 0);
            perror("execlp");
            exit(1);
        }
    }
}
```

多进程并发：server1a.c(1)

```
int main(int argc, char *argv[])
{
    struct sockaddr_in peer;
    unsigned char buf[8192];
    int nbyte, i,  sock,  name_len = sizeof(peer);
    sock = strtol(argv[1], 0, 0);
    getpeername(sock, &peer, &name_len);
    for (;;) {
        nbyte = read(sock, buf, sizeof(buf));
        printf("%s:%d ", inet_ntoa(peer.sin_addr), ntohs(peer.sin_port));
        if (nbyte < 0) {
            perror("Receiving packet");
            exit(1);
        } else if (nbyte == 0) {
            printf("*** Disconnected.\n");
            close(sock);
            exit(0);
        }
        for (i = 0; i < nbyte; i++) printf("%c", buf[i]);
    }
}
```

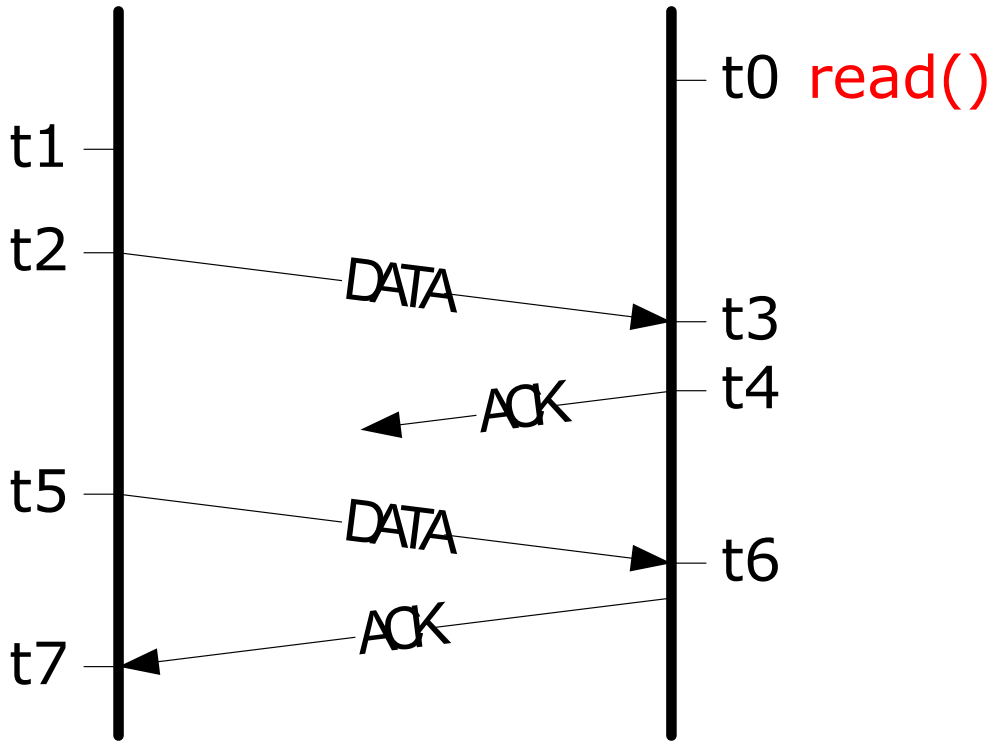
socket系统调用

- **socket**
  - 创建文件描述符socket，端点名未指定
- **bind**
  - 设定本地端点名，也可以用在客户端程序
- **listen**
  - 开始监听到达的连接请求
- **accept**
  - 接受一个连接请求
- **connect**
  - 建立连接，设定远端端点名
- **close**
  - 关闭连接，释放文件描述符

read/write系统调用的语义(1)

read/write与TCP通信的时序

例：主机A用write()通过TCP连接向B发送数据，B接收数据用read()



t0:B开始read()

t1:A调用write()，TCP发送缓冲区有空闲，数据拷贝至发送缓冲区

t2: A将数据发往B

t3: B收到数据后，校验和正确

t4: B向主机A发ACK，ACK途中丢失

t5: A超时自动重发数据

t6: B收到重复的数据后扔掉，回送ACK

t7: A收到ACK，将发送缓冲区的数据清除

read/write系统调用的语义(2)

- write在t1返回，read在t3返回

read/write与TCP通信故障和流控

- 流控问题
- 断线
- 对方重启动
- Keepalive(默认两小时)
- getsockopt/setsockopt可以设置保活间隔，重传次数，重传时间

“粘连问题”

read/write与UDP通信

- 网络故障
- 没有数据粘连
- 没有流控功能
- 不可靠

端点名相关的系统调用

- **getpeername获取对方的端点名**

  getpeername(int *sockfd*, struct sockaddr **name*, int **namelen*);

- **getsockname获取本地的端点名**

  getsockname(int *sockfd*, struct sockaddr **name*, int **namelen*);

read/write的其他版本

int recv(int *sockfd*, void **buf*, int *nbyte*, int *flags*);

int recvfrom(int *sockfd*, void **buf*, int *nbyte*, int *flags*, struct sockaddr **from*, int **fromlen*);

int send(int *sockfd*, void **buf*, int *nbyte*, int *flags*);

int sendto(int *sockfd*, void **buf*, int *nbyte*, int *flags*, struct sockaddr **to*, int *tolen*);

recvfrom/sendto可以指定对方的端点名，常用于UDP

Winsock只能用recv/send不可用read/write

shutdown系统调用(1)

- int shutdown(int *sockfd*, int *howto*);
  - 禁止发送或接收。socket提供全双工通信，两个方向上都可以收发数据，shutdown提供了对于一个方向的通信控制
- 参数*howto*取值
  - SHUT_RD：不能再接收数据，随后read均返回0
  - SHUT_WR：不能再发送数据，本方向再次write会导致SIGPIPE信号
  - SHUT_RDWR：禁止这个*sockfd*上的任何收发

shutdown系统调用(2)

- shutdown是通用的套接字上的操作
  - 执行后对通信的影响，会与具体的通信协议相关
  - TCP协议
    - 允许关闭发送方向的半个连接
    - 没有一种机制让对方关闭它的发送，但TCP协议的流量控制机制，可以通知对方自己的接收窗口为0，对方的write会继续，并将数据堆积在发送缓冲区
  - UDP协议
    - UDP关闭接收方向内核仅记下一个标记，不再提供数据，但无法阻止对方的发送而导致的网络上数据
    - 即使套接字关闭也不影响对方发出无人接收的数据报

socket控制

Socket控制

int getsockopt(int *sokfd*, int *level*, int *optname*, void **optval*, int **optlen*);
int setsockopt(int *sockfd*, int *level*, int *optname*, void **optval*, int *optlen*);
int ioctl(int *fd*, int *cmd*, void **arg*);

无阻塞I/O

```
#include <sys/fcntl.h>
int flags;
flags = fcntl(fd, F_GETFL, 0);
fcntl(fd, F_SETFL, flags | O_NDELAY);
```

- 发送缓冲区满，write立即以-1返回，errno置为EWOULDBLOCK
- 发送缓冲区半满，write返回实际发送的字节数
- 接收缓冲区空，read立即以-1返回，errno置为EWOULDBLOCK

单进程并发处理

select：多路I/O

- 引入select系统调用的原因
  - 使得用户进程可同时等待多个事件发生
  - 用户进程告知内核多个事件，某一个或多个事件发生时select返回，否则，进程睡眠等待
- int select(int *maxfdp1*, fd_set **rfds*, fd_set **wfds*, fd_set **efds*, struct timeval **timeout*);
  - 例如：告知内核在*rfds*集合{4,5,7}中的任何文件描述符“读准备好”，或在*wfds*集合{3,7}中的任何文件描述符“写准备好”，或在*efds*集合{4,5,8}中的任何文件描述符有“异常情况”发生
  - 集合参数是传入传出型，select返回后会被修改，只有准备好文件描述符，仍出现在集合中
  - 集合参数允许传NULL，表示不关心这方面事件

select：“准备好”

- **什么叫“准备好”**
  - *rfds*中某文件描述符的read不会阻塞
  - *wfds*中某文件描述符的write不会阻塞
  - *efds*中某文件描述符发生了异常情况
    - TCP协议，只有加急数据到达才算“异常情况”
    - 对方连接关闭或网络故障，不算“异常情况”
- **“准备好”后可以进行的操作**
  - 当“读准备好”时，调用read会立刻返回-1/0/字节数
  - 当“写准备好”时，调用write可以写多少字节？
    - >=1个字节
    - “无阻塞I/O”方式

集合操作

预定义数据类型fd_set（在C语言头文件定义）

- **void FD_ZERO(fd_set *_fds_);**

  将*fds*清零：将集合*fds*设置为“空集”

- **void FD_SET(int _fd_, fd_set *_fds_);**

  向集合*fds*中加入一个元素*fd*

- **void FD_CLR(int _fd_, fd_set *_fds_);**

  从集合*fds*中删除一个元素*fd*

- **int FD_ISSET(int _fd_, fd_set *_fds_);**

  判断元素*fd*是否在集合*fds*内

select：时间

- 结构体定义

```
struct timeval {
    long tv_sec;   /* 秒 */
    long tv_usec; /* 微秒 */
};
```

- **select的最后一个参数*timeout***
  - 定时值不为0：secect在某一个描述符I/O就绪时立即返回；否则等待但不超过*timeout*规定的时限
    - 尽管*timeout*可指定微秒级精度的时间段，依赖于硬件和软件的设定，实际实现一般是10毫秒级别
  - 定时值为0：select立即返回(无阻塞方式查询)
  - 空指针NULL：select等待到至少有一个文件描述符准备好后才返回，否则无限期地等下去

单进程并发：server2.c(1)

```
#define PORT_NO 12345
int main(void)
{
    int admin_sock, data_sock, ret, maxfdp1, fd;
    struct sockaddr_in name;
    fd_set fds, rfds;
    admin_sock = socket(AF_INET, SOCK_STREAM, 0);
    name.sin_family = AF_INET;
    name.sin_addr.s_addr = INADDR_ANY;
    name.sin_port = htons(PORT_NO);
    if (bind(admin_sock, &name, sizeof(name)) < 0) {
        perror("bind");
        exit(1);
    }
    listen(admin_sock, 5);
    printf("ready\n");
    maxfdp1 = admin_sock + 1;
    FD_ZERO(&fds);
    FD_SET(admin_sock, &fds);
```

单进程并发：server2.c(2)

```
for(;;) {
        memcpy(&rfds, &fds, sizeof(fds));
        ret = select(n, &rfds, 0, 0, 0);
        if (FD_ISSET(admin_sock, &rfds)) {
            data_sock = accept(admin_sock, 0, 0);
            if (data_sock < 0) . . .;
            FD_SET(data_sock, &fds);
            if (n <= data_sock) n = data_sock + 1;
        }
        for (fd = 0; fd < n; fd++) {
            if (fd != admin_sock && FD_ISSET(fd, &rfds)) {
                if (receive_data(fd) == 0) {
                    close(fd);
                    FD_CLR(fd, &fds);
                }
            }
        }
    }
}
```

单进程并发：server2.c(3)

```
int receive_data(int sock)
{
    unsigned char rbuf[8192];
    struct sockaddr_in peer;
    int i, nbyte, name_len = sizeof(peer);

    nbyte = recvfrom(sock, rbuf, SIZE, 0, &peer, &name_len);
    if (nbyte < 0) {
        perror("receiving stream packet");
        return 0;
    }
    printf("%s:%d ", inet_ntoa(peer.sin_addr), ntohs(peer.sin_port));
    if (nbyte == 0) {
        printf("*** Disconnected.\n");
        return 0;
    }
    for (i = 0; i < nbyte; i++) printf("%c", rbuf[i]);
    return 1;
}
```

UDP通信

客户端udpclient.c

```
#define PORT_NO 12345
int main(int argc, char *argv[])
{
    int sock, len;
    struct sockaddr_in name;
    char sbuf[8192];
    if (argc <2 ) . . . ;
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    name.sin_family = AF_INET;
    name.sin_addr.s_addr = htonl(inet_network(argv[1]));
    name.sin_port = htons(PORT_NO);
    connect(sock, &name, sizeof(name));
   for(;;) {
        if (fgets(sbuf, sizeof sbuf, stdin) == NULL) break;
        len = write(sock, sbuf, strlen(sbuf));
        if (len < 0) ...;
        printf("send %d bytes\n", len);
    }
    close(sock);
}
```

客户端程序udpclient.c(2)

- **connect**
  - 不产生网络流量，内核记下远端端点名
  - 之前未用bind指定本地端点名，系统自动分配本地端点名
- **write**
  - 使用前面的connect调用指定的端点名
  - UDP不是面向连接的协议，可在sendto参数中指定对方端点名，而且允许对方端点名不同
  - 每次都使用sendto发送数据，前面的connect调用没必要了
  - connect/第一次sendto可使得socket获得系统动态分配的本地端点名，未获得本地端点名之前不该执行read或recv以及recvfrom

服务端程序udpserver.c

```
int main(void)
{
    int sock, len;
    struct sockaddr_in name;
    unsigned char buf[8192];
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    name.sin_family = AF_INET;
    name.sin_addr.s_addr = INADDR_ANY;
    name.sin_port = htons(12345);
    if (bind(sock, &name, sizeof(name)) < 0) {
        perror("binding socket");
        exit(1);
    }
    for (;;) {
         len = read(sock, buf, sizeof buf);
         if (len > 0) printf("Receive %d bytes\n", len);
     }
 }
```

UDP通信程序

- **接收**
  - 没有数据到达时，read调用会使得进程睡眠等待
  - 一般需区分数据来自何处，常用recvfrom获得对方的端点名
- **发送**
  - 服务器端发送数据常用sendto，指定远端端点名
  - 对接收来的数据作应答，sendto引用的对方端点名利用recvfrom返回得到的端点名
- **select定时**
  - select可实现同时等待两个事件：收到数据和定时器超时
  - 用time(0)或者gettimeofday()获得时间坐标，计算时间间隔决定是否执行超时后的动作
- **死锁问题**

复习

- 协议栈实现：用户态还是核心态
- 三类特殊文件：管道、终端、socket
- Socket机制不仅用于TCP/IP
- TCP/UDP服务模型
- 网络字节顺序问题
- TCP客户服务器
- 端点名的概念
- 客户端程序的几个重要调用以及进程状态
- 服务器端程序
- 服务器程序的几个调用
- 多进程并发的文件描述符问题
- inetd的实现方法
- TCP服务端设多个socket原因
- read/write与进程状态，以及与TCP协议操作的时序关系
- Socket感知网络故障的手段与TCP协议的对应关系
- 获取端点名系统调用
- read/write的其他版本
- shutdown调用：功能以及与TCP协议的关系
- 无阻塞I/O
- 单进程并发处理方法
- 准确理解select调用：什么叫“准备好“
- 集合操作
- select的时间参数
- UDP通信