计算机网络课实验（6）

1 实验名称

利用 Socket API 实现网上点对点通信。

2 实验要求

在 Windows 或 Linux 操作系统（也可以将客户端部署在 Android、iOS 或 WinPhone 手机）下，分别基于 TCP 和 UDP 协议，利用 Socket API 实现网上点对点通信。

需要实现两种功能！程序一、二可以分开写，也可写在一起。

程序一“基于 TCP 的可靠文件传输”，功能包括：

在客户端，用户选择本地的某个文件，并发送到服务器端。

在服务器端，接收客户端传输的数据流，并按 IP 地址保存在服务器端（文件名重复的，可以覆盖）。

如果传输过程中服务器端发现客户端断开，服务器端应删除文件，并在屏幕上提示，如“IP：1.2.3.4 发来 abcd.txt 文件过程中失去连接。”。如果客户端发现服务器端不工作，客户端应有提示“服务器 1.2.3.5:62345 失去连接”。

程序二“基于 UDP 的不可靠文件传输”，功能同上，但不能使用 TCP 协议进行传输。考虑如果传输过程中服务器端、客户端如何发现断开。

要求：

（1）开发环境自选。

（2）不要选用公用端口号（0~49151）。

（3）注意对出错进行处理，在传输过程中可以使用任务管理器终止服务器或客户端模拟实现。

用 C/C++实现，开发环境和操作系统自选。建议不要选用公用端口号（0~49151）。

3 思考题

1、一个应用可以对应几个IP？几个端口？
2、一个端口可以给几个应用使用？
3、能不能随意使用知名端口？为什么？举例。
4、随机问题*1，根据你的代码或实现方式进行提问。

4 实验提交文件

报告和源码文件一并打包提交，命名格式为：“E6+学号+姓名”。

报告

对于结果图片，可以根据验收项，将对应结果截图。其余要求照旧。

源码

只需提交核心文件

5 附录： 基于 Socket 的 UDP 和 TCP 编程介绍

一、概述

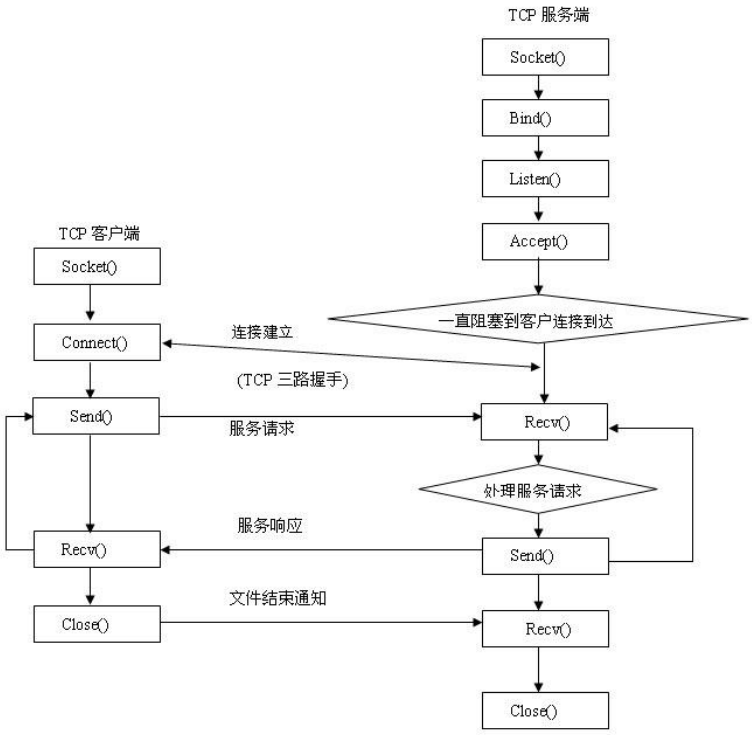
TCP(传输控制协议)和 UDP(用户数据报协议)是网络体系结构 TCP/IP 模型中传输层一层中的两个不同的通信协议。

TCP：传输控制协议，一种面向连接的协议，给用户进程提供可靠的全双工的字节流，TCP 套接口是字节流套接口(stream socket)的一种。

UDP：用户数据报协议。UDP 是一种无连接协议。UDP 套接口是数据报套接口(datagram socket)的一种。

二、TCP 和 UDP 介绍

1) 基本 TCP 客户—服务器程序设计基本框架

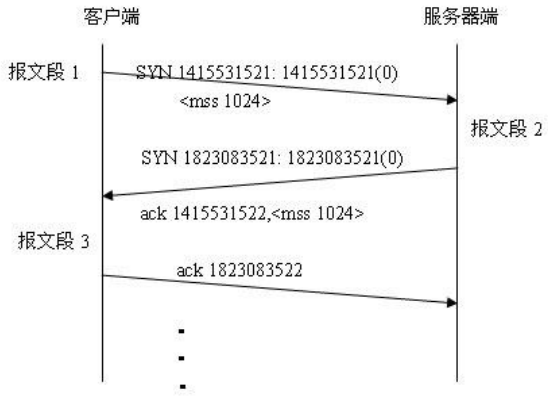


说明：（三路握手）

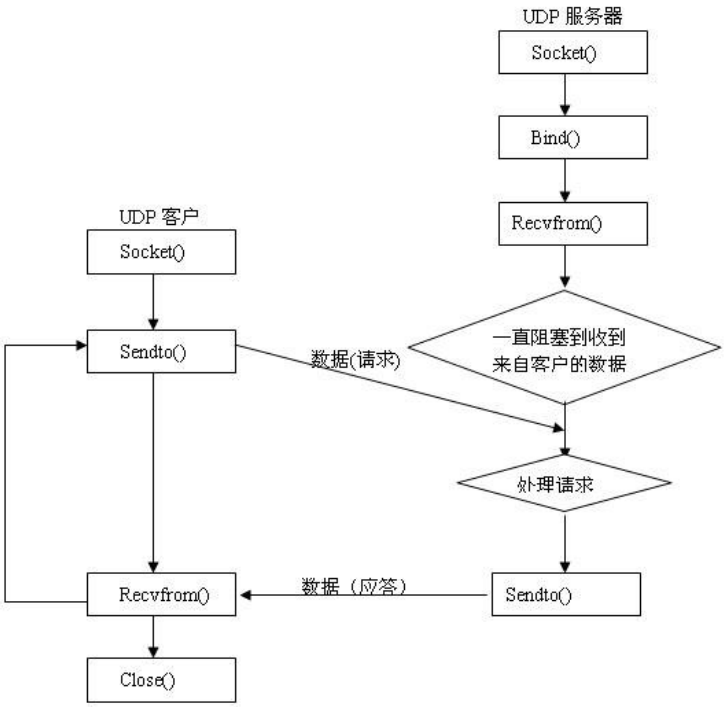
1.客户端发送一个 SYN 段（同步序号）指明客户打算连接的服务器端口，以及初始化序号(ISN) 。

2.服务器发回包含服务器的初始序号的 SYN 报文段作为应答。同时，将确认序号(ACK)设置为客户的 ISN 加 1 以对客户的 SYN 报文段进行确认。一个 SYN 将占用一个序号。

3.客户必须将确认序号设置为服务器的 ISN 加 1 以对服务器的 SYN 报文段进行确认。



2) 基本 UDP 客户—服务器程序设计基本框架流程图



3) UDP 和 TCP 的对比：

从上面的流程图比较我们可以很明显的看出 UDP 没有三次握手过程。

简单点说。UDP 处理的细节比 TCP 少。UDP 不能保证消息被传送到（它也报告消息没有传送到）目的地。UDP 也不保证数据包的传送顺序。UDP 把数据发出去后只能希望它能够抵达目的地。

TCP 优缺点：

优点：

1．TCP 提供以认可的方式显式地创建和终止连接。

2．TCP 保证可靠的、顺序的（数据包以发送的顺序接收）以及不会重复的数据传输。

3．TCP 处理流控制。

4．允许数据优先

5．如果数据没有传送到，则 TCP 套接口返回一个出错状态条件。

6．TCP 通过保持连续并将数据块分成更小的分片来处理大数据块。—无需程序员知道

缺点： TCP 在转移数据时必须创建（并保持）一个连接。这个连接给通信进程增加了开销，让它比 UDP 速度要慢。

UDP 优缺点：

1．UDP 不要求保持一个连接

2．UDP 没有因接收方认可收到数据包（或者当数据包没有正确抵达而自动重传）而带来的开销。

3．设计 UDP 的目的是用于短应用和控制消息

4．在一个数据包连接一个数据包的基础上，UDP 要求的网络带宽比 TCP 更小。

三、Socket 编程

Socket 接口是 TCP/IP 网络的 API，Socket 接口定义了许多函数或例程，程序员可以用它们来开发 TCP/IP 网络上的应用程序。要学 Internet 上的 TCP/IP 网络编程，必须理解 Socket 接口。

Socket接口设计者最先是将接口放在Unix操作系统里面的。如果了解Unix系统的输入和输出的话，就很容易了解Socket了。网络的Socket数据传输是一种特殊的I/O，Socket也是一种文件描述符。Socket也具有一个类似于打开文件的函数调用Socket()，该函数返回一个整型的Socket描述符，随后的连接建立、数据传输等操作都是通过该Socket实现的。常用的Socket类型有两种：流式Socket（SOCK_STREAM）和数据报式Socket（SOCK_DGRAM）。流式是一种面向连接的Socket，针对于面向连接的TCP服务应用；数据报式Socket是一种无连接的Socket，对应于无连接的UDP服务应用。

1、socket调用库函数主要有：

创建套接字

Socket(af,type,protocol)

建立地址和套接字的联系

bind(sockid, local addr, addrlen)

服务器端侦听客户端的请求

listen(Sockid ,quenlen)

建立服务器/客户端的连接 (面向连接TCP)

客户端请求连接

Connect(sockid, destaddr, addrlen)

服务器端等待从编号为Sockid的Socket上接收客户连接请求

newsockid=accept(Sockid，Clientaddr, paddrlen)

发送/接收数据

面向连接：send(sockid, buff, bufflen)

recv( )

面向无连接：sendto(sockid,buff,…,addrlen)

recvfrom( )

释放套接字

close(sockid)

2、TCP/IP 应用编程接口（API）

服务器的工作流程：首先调用 socket 函数创建一个 Socket，然后调用 bind 函数将其与本机地址以及一个本地端口号绑定，然后调用 listen 在相应的 socket 上监听，当 accpet 接收到一个连接服务请求时，将生成一个新的 socket。服务器显示该客户机的 IP 地址，并通过新的 socket 向客户端发送字符串" hi,I am server!"。最后关闭该 socket。

```
main()
{
    int sock_fd, client_fd; /*sock_fd：监听socket；client_fd：数据传输socket */
    struct sockaddr_in ser_addr; /* 本机地址信息 */
    struct sockaddr_in cli_addr; /* 客户端地址信息 */
    char msg[MAX_MSG_SIZE];/* 缓冲区*/
    ser_sockfd = socket(AF_INET, SOCK_STREAM, 0);/*创建连接的SOCKET */
    if (ser_sockfd < 0)
    {/*创建失败 */
        fprintf(stderr, "socker Error:%s\n", strerror(errno));
        exit(1);
```

```
    }
    /* 初始化服务器地址*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&ser_addr, addrlen);
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    ser_addr.sin_port = htons(SERVER_PORT);
    if (bind(ser_sockfd, (struct sockaddr*)&ser_addr, sizeof(struct sockaddr_in))
< 0)
    { /*绑定失败 */
        fprintf(stderr, "Bind Error:%s\n", strerror(errno));
        exit(1);
    }
    /*侦听客户端请求*/
    if (listen(ser_sockfd, BACKLOG) < 0)
    {
        fprintf(stderr, "Listen Error:%s\n", strerror(errno));
        close(ser_sockfd);
        exit(1);
    }
    while (1)
    {/* 等待接收客户连接请求*/
        cli_sockfd = accept(ser_sockfd, (struct sockaddr*) &        cli_addr,
&addrlen);
        if (cli_sockfd <= 0)
        {
            fprintf(stderr, "Accept Error:%s\n", strerror(errno));
        }
        else
        {/*开始服务*/
            recv(cli_addr, msg, MAX_MSG_SIZE, 0); /* 接受数据*/
            printf("received a connection from %sn", inet_ntoa(cli_addr.sin_addr));
            printf("%s\n", msg);/*在屏幕上打印出来 */
            strcpy(msg, "hi,I am server!");
            send(cli_addr, msg, sizeof(msg), 0); /*发送的数据*/
            close(cli_addr);
        }
    }
    close(ser_sockfd);
}
```

客户端的工作流程：首先调用 socket 函数创建一个 Socket，然后调用 bind 函数将其与本机地址以及一个本地端口号绑定，请求连接服务器，通过新的 socket 向客户端发送字符串" hi,I am client!"。最后关闭该 socket。

```
main()
{
    int cli_sockfd;/*客户端SOCKET */
    int addrlen;
    char seraddr[14];
    struct sockaddr_in ser_addr,/* 服务器的地址*/
        cli_addr;/* 客户端的地址*/
    char msg[MAX_MSG_SIZE];/* 缓冲区*/
    GetServerAddr(seraddr);
    cli_sockfd = socket(AF_INET, SOCK_STREAM, 0);/*创建连接的SOCKET */
    if (ser_sockfd < 0)
    {/*创建失败 */
        fprintf(stderr, "socker Error:%s\n", strerror(errno));
        exit(1);
    }
    /* 初始化客户端地址*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&ser_addr, addrlen);
    cli_addr.sin_family = AF_INET;
    cli_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    cli_addr.sin_port = 0;
    if (bind(cli_sockfd, (struct sockaddr*)&cli_addr, addrlen) < 0)
    {
        /*棒定失败 */
        fprintf(stderr, "Bind Error:%s\n", strerror(errno));
        exit(1);
    }
    /* 初始化服务器地址*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&ser_addr, addrlen);
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_addr.s_addr = inet_addr(seraddr);
    ser_addr.sin_port = htons(SERVER_PORT);
    if (connect(cli_sockfd, (struct sockaddr*)&ser_addr, &addrlen) != 0)/*请求连
接*/
    {
```

```
        /*连接失败 */
        fprintf(stderr, "Connect Error:%s\n", strerror(errno));
        close(cli_sockfd);
        exit(1);
    }
    strcpy(msg, "hi,I am client!");
    send(sockfd, msg, sizeof(msg), 0);/*发送数据*/
    recv(sockfd, msg, MAX_MSG_SIZE, 0); /* 接受数据*/
    printf("%s\n", msg);/*在屏幕上打印出来 */
    close(cli_sockfd);
}
```

3、UDP/IP 应用编程接口（API）

服务器的工作流程：首先调用 socket 函数创建一个 Socket，然后调用 bind 函数将其与本机地址以及一个本地端口号绑定，接收到一个客户端时，服务器显示该客户端的 IP 地址，并将字串返回给客户端。

```
int main(int argc, char **argv)
{
    int ser_sockfd;
    int len;
    //int addrlen;
    socklen_t addrlen;
    char seraddr[100];
    struct sockaddr_in ser_addr;
    /*建立socket*/
    ser_sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (ser_sockfd < 0)
    {
        printf("I cannot socket success\n");
        return 1;
    }
    /*填写sockaddr_in 结构*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&ser_addr, addrlen);
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    ser_addr.sin_port = htons(SERVER_PORT);
    /*绑定客户端*/
```

```
    if (bind(ser_sockfd, (struct sockaddr *)&ser_addr, addrlen) < 0)
    {
        printf("connect");
        return 1;
    }
    while (1)
    {
        bzero(seraddr, sizeof(seraddr));
        len = recvfrom(ser_sockfd, seraddr, sizeof(seraddr), 0, (struct
sockaddr*)&ser_addr, &addrlen);
        /*显示client端的网络地址*/
        printf("receive from %s\n", inet_ntoa(ser_addr.sin_addr));
        /*显示客户端发来的字串*/
        printf("recevce:%s", seraddr);
        /*将字串返回给client端*/
        sendto(ser_sockfd, seraddr, len, 0, (struct sockaddr*)&ser_addr, addrlen);
    }
}
```

客户端的工作流程：首先调用 socket 函数创建一个 Socket，填写服务器地址及端口号，从标准输入设备中取得字符串，将字符串传送给服务器端，并接收服务器端返回的字符串。最后关闭该 socket。

```
int GetServerAddr(char * addrname)
{
    printf("please input server addr:");
    scanf("%s", addrname);
    return 1;
}
int main(int argc, char **argv)
{
    int cli_sockfd;
    int len;
    socklen_t addrlen;
    char seraddr[14];
    struct sockaddr_in cli_addr;
    char buffer[256];
    GetServerAddr(seraddr);
    /* 建立socket*/
    cli_sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

```
    if (cli_sockfd < 0)
    {
        printf("I cannot socket success\n");
        return 1;
    }
    /* 填写sockaddr_in*/
    addrlen = sizeof(struct sockaddr_in);
    bzero(&cli_addr, addrlen);
    cli_addr.sin_family = AF_INET;
    cli_addr.sin_addr.s_addr = inet_addr(seraddr);
    //cli_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    cli_addr.sin_port = htons(SERVER_PORT);
    bzero(buffer, sizeof(buffer));
    /* 从标准输入设备取得字符串*/
    len = read(STDIN_FILENO, buffer, sizeof(buffer));
    /* 将字符串传送给server端*/
    sendto(cli_sockfd, buffer, len, 0, (struct sockaddr*)&cli_addr, addrlen);
    /* 接收server端返回的字符串*/
    len = recvfrom(cli_sockfd, buffer, sizeof(buffer), 0, (struct
sockaddr*)&cli_addr, &addrlen);
    //printf("receive from %s\n",inet_ntoa(cli_addr.sin_addr));
    printf("receive: %s", buffer);
    close(cli_sockfd);
}
```