IPv6 网络基础编程

一、实验目的

学习网络套接口（socket）编程，掌握 Linux 操作系统下使用 TCP 协议进行通信的 IPv6 网络应用程序基本实现框架，加深对 IPv6 协议的理解。

二、预备工作和实验要求

将纯 IPv4 网络应用程序移植到纯 IPv6 环境下并没有多大困难；对于典型的 C/S 结构程序，我们只需对客户端和服务器端源代码进行简单修改，然后重新编译它们。本次试验就是编写 IPv6 客户端和 IPv6 服务器端使用 TCP 协议进行通信的应用程序，从 IPv6 的角度来学习 socket 编程知识。

本次实验内容相对简单，涉及的概念也不多。上机实验之前可先复习教材 4.3.8 节。查阅有关编程资料，具备一些初步的 C 语言网络编程知识即可。

三、实验内容

1、编写使用 TCP 协议的 IPv6 客户端程序

使用 vim 或其它编辑器，在自己作实验用的机器上输入以下源程序。

```
#include <stdio.h>
#include <netinet/in.h>
#define MAXLINE 1024

int main (int argc, char **argv)
{
    int sockfd, n;
    char recvline [MAXLINE+1];
    char mesg[] = "Hello World!";
    struct sockaddr_in6 servaddr;

    if (argc!=2) {
        printf ("usage: %s <IPv6_address>\n", argv[0]);
        exit (1);
    }
    if ((sockfd = socket (AF_INET6, SOCK_STREAM, 0)) < 0) {
```

```
        perror ("socket");
        exit (2);
    }
    bzero (&servaddr, sizeof (servaddr));
    servaddr.sin6_family = AF_INET6;
    servaddr.sin6_port = htons (7);
    if (inet_pton (AF_INET6, argv[1], &servaddr.sin6_addr) <= 0) {
        printf ("inet_pton error for %s\n", argv[1]);
        exit (3);
    }
    if (connect (sockfd, (struct sockaddr *)&servaddr, sizeof (servaddr)) < 0) {
        perror ("connect");
        exit (4);
    }
    if (write (sockfd, mesg, strlen(mesg)) < 0) {
        perror ("write");
        exit (5);
    }

    n = read (sockfd, recvline, MAXLINE);
    if (n < 0) {
        perror ("read");
        exit (6);
    }
    recvline [n] = '\0';
    if (puts (recvline) == EOF) {
        perror ("puts");
        exit (7);
    }
    close (sockfd);
    return 0;
}
```

编译后运行./a.out 3ffe:3216:2101:2100::3，记录输出结果。这是一个何种网络应用的客户端程序？试着把"Hello World!"换成其它的字符串再编译运行，用实验来证实或修正你的判断。

接下来我们再利用这个 IPv6 客户端程序来完成一个有趣的操作：从 www 服务器获取其当前系统时间（GMT 时间，比中国北京时间慢 8 个小时）。其基本思想是利用 HTTP 协议中的 HEAD 指令。对程序只需要作两处修改，如下所示：

①char mesg[] = "Hello World!";改换成 char mesg[] = "HEAD / HTTP\n\n";

②servaddr.sin6_port = htons (7);改换成 servaddr.sin6_port = htons (80);

重新编译后运行./a.out 3ffe:3216:2101:2100::5，记录输出结果。从一定意义上来看，我们仅仅作了很小的更动就获得了一个日期服务（daytime）的客户端程序（尽管我们并未连接到一个真正提供 daytime 服务的网络主机上）。

注意：①HTTP 协议的 HEAD 指令中，“/”号前后各有一个空格。②由于校园网内 IPv6 系统的域名解析有时不够稳定，以上的 TCP 连接速度可能会较慢，发生此现象时请同学们稍微等待。③本次实验给出的程序是在一台安装 Mandrake Linux 6.1 的 PC 机上调试通过的，在不同的 Linux 环境下可能要做个别调整。

2、编写使用 TCP 协议的 IPv6 服务器端程序

使用 vim 或其它编辑器，在自己作实验用的机器上输入以下源程序。

```
#include <netdb.h>
#include <netinet/in.h>
#define MAXLINE 1024

char *ipv6addr (struct sockaddr_in6 *sa)
{
    static char string[40];
    unsigned char *p, *q = string;
    int i = 1;
    for (p = sa->sin6_addr.in6_u.u6_addr8; i<=8; i++) {
        sprintf (q, "%02x", *p++); q+=2;
        sprintf (q, "%02x", *p++); q+=2;
        sprintf (q++, ":");
    }
    return string;
}

int tcp_listen (const char *host, const char *serv, socklen_t *addrlenp)
{
    int listenfd, n;
    const int on = 1;
    struct addrinfo hints, *res, *ressave;

    bzero (&hints, sizeof (struct addrinfo));
    hints.ai_flags = AI_PASSIVE;
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    if ((n = getaddrinfo (host, serv, &hints, &res)) != 0) {
        perror ("getaddrinfo in tcp_listen");
        exit (1);
    }

    ressave = res;
    do {
        listenfd = socket (res->ai_family, res->ai_socktype, res->ai_protocol);
```

```
        if (listenfd < 0)
            continue; //let's try next one
        setsockopt (listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof (on));
        if (bind (listenfd, res->ai_addr, res->ai_addrlen) == 0)
            break; //success
        close (listenfd); //close and try next one
    } while ((res = res->ai_next) != NULL);

    if (res == NULL) { //even the last try in socket() or bind() failed, faint
        perror ("exception in tcp_listen");
        exit (1);
    }
    if (listen (listenfd, 1024) < 0) {
        perror ("listen in tcp_listen");
        exit (1);
    }
    if (addrlenp) *addrlenp = res->ai_addrlen; //size of protocol address
    freeaddrinfo (ressave);
    return (listenfd);
}

int main (int argc, char** argv)
{
    int listenfd, connfd;
    socklen_t addrlen;
    char buff[MAXLINE];
    time_t ticks;
    struct sockaddr_in6 *cliaddr;

    if (argc == 3)
        listenfd = tcp_listen (argv[1], argv[2], &addrlen);
    else {
        printf ("usage: %s <Host_IP> <TCP_Port>\n", argv[0]);
        exit (1);
    }
    if ((cliaddr = malloc (addrlen)) == NULL) {
        perror ("malloc");
        exit (1);
    }
    while (1) {
        connfd = accept (listenfd, cliaddr, &addrlen);
        printf ("connection from %.39s\n", ipv6addr(cliaddr));
        ticks = time (NULL);
        snprintf (buff, sizeof (buff), "%.24s\r\n", ctime (&ticks));
```

```
        write (connfd, buff, strlen (buff));
        close (connfd);
    }
}
```

假定你做实验用的机器配置的 IPv6 地址是 ADDRESS，IPv4 地址是 address。编译以上程序通过后在 Alt+F1 对应的控制台运行./a.out ::1 2000，在 Alt+F2 对应的控制台依次进行三个 telnet 操作：①从本机 telnet ::1 2000，②登录到实验室网关后再 telnet ADDRESS 2000，③从网关 telnet address 2000，记录结果。这是一个何种网络应用的服务器端程序？思考为什么三次 telnet 操作的实验现象不同。

在 Alt+F1 对应的控制台运行./a.out ADDRESS 2000，然后重复上面①②③的操作并记录。最后，在 Alt+F1 对应的控制台运行./a.out ::0 2000，然后重复上面①②③的操作并记录。注意每次运行 a.out 前应该使用 Ctrl+c 终止上一次的进程。

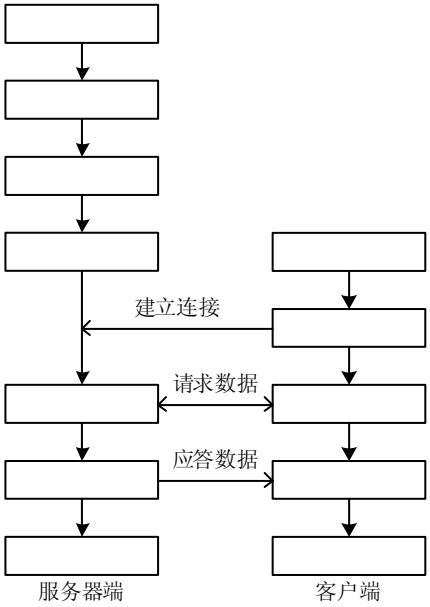
在以上测试的各个输出，网关使用的地址是什么（即服务器程序所打印出的 connection from 是什么地址）？操作③中网关使用 IPv4 地址与你所在机器通信，那么这时打印出的地址是 IPv4-compatible 地址还是 IPv4-mapped 地址？

四、习题

1、TCP/IP 协议栈由哪四层组成？网络套口 socket 处于这个模型中的什么位置，起何作用或有何意义？

2、实验内容 2 中给出的 char *ipv6addr (struct sockaddr_in6 *sa)这个函数作用是什么？它在实现上稍微显得有些古怪（例如赋值语句 p = sa->sin6_addr.in6_u.u6_addr8）。请到/usr/include/linux/in6.h（或相应的头文件）中查看 struct sockaddr_in6 及与其有关的各个数据结构的 C 语言定义，记录在 Linux 操作系统下 IPv6 的 128 比特地址是以何种方式实现存储的。

3、对第 1 题中的 ipv6addr 函数进行修改（保持函数原型不变），使得 main 函数 while(1)循环中的 printf 能同时打印出客户端使用的 TCP 端口号。（提示:①注意网络字节顺序与主机字节顺序的转换问题，②ipv6addr 函数中的静态字符数组 string 需再增加长度以容纳下各种可能的端口号，③main 函数 while (1)循环中 printf 的调用参数%.39s 需按照②中所设相应扩大）。



习题 5 流程图

4、实验中出现了一个“::0”的地址，结合实验内容 2 的结果说明“::0”在 socket 编程中指代什么含义。以此为线索，从网络安全的角度分析服务器端应用程序应该以什么策略选择侦听端口的绑定方式。

5、本实验和上一实验都涉及了 C/S 模型 TCP 网络通信编程，以下就是一个典型的 C/S 模型 TCP 通信程序流程图。请将 socket()、write()、close()、read()、listen()、bind()、connect()、accept()这八个基本函数填入流程图中。

五、附录：libnet 简介

基于 socket 的编程已成为当今网络编程的主流，这种编程思想将网络通讯当作“文件”描述符进行处理，对这个“网络文件”即 socket 的操作从编程者的角度来讲与普通的文件操作（如打开、读写、关闭等）大同小异，从而极大地简化了网络程序的开发过程。

另一方面，为了简化网络程序的编写过程（尤其是一些特殊的需要涉及低层网络功能的程序），提高软件的性能和健壮性，同时使代码更易移植与重用，最好的方法就是将最常用和最繁复的过程函数，如监听套接口的打开/关闭、数据包构造/发送/接收等，封装成库，以 API 的方式提供给开发人员使用。

在 Unix 类操作系统的网络工具开发中，目前最为流行的 C API 库有 libnet、libpcap、libnids 和 libicmp 等。它们分别从不同的层次提供了不同的功能函数，使网络开发人员能够忽略网络最底层细节的实现，从而专注于程序本身具体功能的设计与开发。我们在这里要介绍的 libnet 提供的接口函数主要实现和封装了数据包的构造和发送过程，可以在绝大多数 UNIX 类系统中使用。利用这些 C 函数库的接口，网络安全工具开发人员可以很方便地编写出具有结构化强、健壮性好、可移植性高等特点的程序。

libnet 提供的接口函数包含很多种数据包生成器和两种数据包发送器（IP 层和数据链路层）。目前只支持 IPv4，不支持 IPv6。按照 The Libnet Reference Manual 中所述，对 IPv6 的支持已经明确列入 libnet 的开发计划之中。

libnet 提供的接口函数按其作用可分为四类：内存管理（分配和释放）函数、地址解析函数、数据包构造函数、数据包发送函数，其英文术语依次为 Address Resolution Routines、Packet Memory Management Routines、Packet Injection Framework Routines

和 Packet Builder Routines。

利用 libnet 函数库开发应用程序的基本步骤非常简单，典型的为：1、数据包内存初始化；2、网络接口初始化；3、构造所需数据包；4、计算数据包检验和；5、发送数据包；6、关闭网络接口，释放数据包所占用内存。这些 API 函数的名称均以 libnet_开头，所编写出的函数也非常清晰易读。