实验三、嵌入式 Linux 网络通信实验

一、实验目的

1. 掌握 TCP 与 UDP 协议原理
2. 掌握套接字通信原理。
2. 掌握 TCP 套接字服务器端与客户端通信方法。

二、实验基本要求

1. 学习 TCP 与 UDP 协议原理。
2. 掌握 TCP 套接字服务器端与客户端通信方法，实验箱和 Ubuntu 进行 TCP 通信。

三、实验原理

1．TCP 协议与 UDP 协议

TCP 协议（传输控制协议）是面向连接的通信协议，TCP 提供两台计算机之间的可靠无差错的数据传输。应用程序利用 TCP 进行通信时，客户端和服务器端之间建立一个套接字连接，可以把套接字连接想象为一个电话呼叫，只有呼叫成功时，双方就能进行通话。建立连接的两计算机之间就可以把数据当作一个双向字节流进行传输。一般把在最初建立呼叫时，主动呼叫方称为“客户端”，负责监听方称为“服务器端”。

UDP 协议(用户数据报协议)是无连接通信协议，UDP 不保证可靠的数据传输。如果一个主机向另一个主机发送数据，无需建立连接就会直接将数据发出去，而不管另一台主机是否准备接受数据。如果另一个主机收到了数据，它不会向对方发送收到确认信息。这一过程，类似于从邮局发送信件，无法确认收信人一定能收到发出去的信件。

2．套接字概述

通过 IP 地址可以在网络上找到主机，通过端口号找到主机上正在运行的程序（进程）。也就是说，网络通信不能简单地说成是两台计算机之间的通信，而是两台计算机上执行的应用程序间收发数据。套接字就是 IP 地址和端口号的组合，用以表征一个虚拟的文件句柄。

根据网络传输协议类型的不同，套接字分为以下三种类型：

（1）字节流套接字。又称 TCP 套接字，基于 TCP 协议连接和传输方式，能保证数据传输的正确性和顺序性。

（2）数据报套接字。又称 UDP 套接字，基于 UDP 协议的连接和传输方式，它定义了一种无连接的服务，数据通过相互独立的数据报进行传输，并且无需对传输的数据进行确认，传输速度快。

（3）原始套接字。原始套接字允许对底层协议如 IP 或 IMCP 进行直接访问，主要用于对一些协议的开发，构造自己的数据报分组。

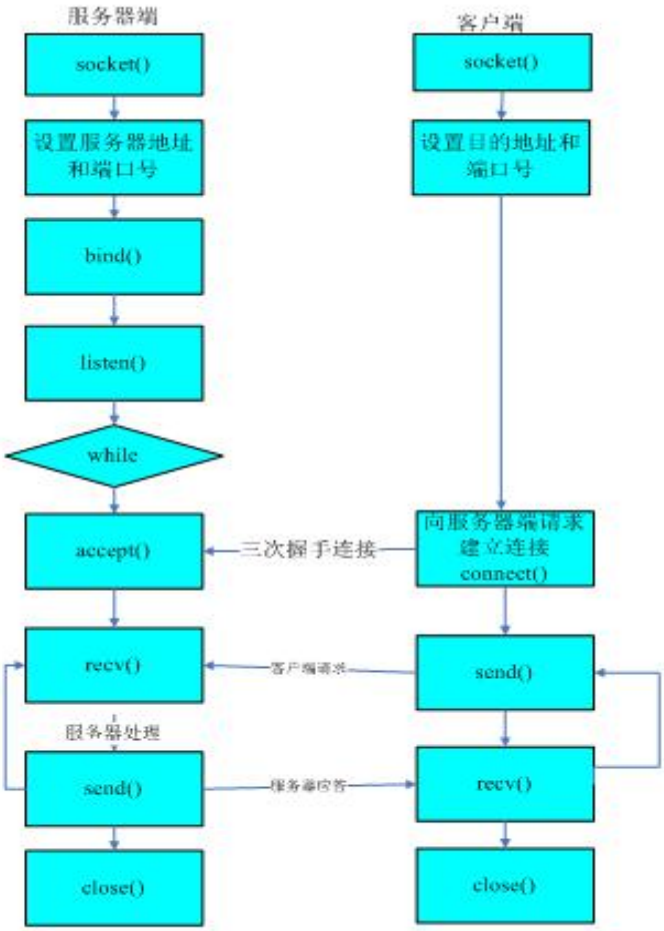
3．TCP 套接字通信步骤

3.1 服务器端

（1）调用 socket()创建套接字，然后初始化 struct sockaddr_in 结构体。
（2）调用 bind 函数()为套接字绑定一个 IP 地址和一个端口号。
（3）调用 listen()函数使套接字成为监听套接字，侦听指定的端口。
（4）调用 accept()函数，使服务器处于阻塞状态，等待接受客户端连接请求。一旦建立连接，将产生新的套接字，此时就有两个套接字了，原来的那个套接字还在监听等待指定的端口，而新产生的套接字则准备发送或接受数据。
（5）利用 send/sendto 和 recv/recvfrom 进行数据传输。当然也可以调用 write 或 read.
（6）数据传输完毕，关闭套接字。

3.2 客户端

（1）调用 socket()创建套接字，然后初始化 struct sockaddr_in 结构体，注意服务器端和客户端的 struct sockaddr_in 结构体应该一致。
（2）调用 connection()函数与服务器建立连接。
（3）利用 send/sendto 和 recv/recvfrom 进行数据传输. 当然也可以调用 write 或 read.
（4）数据传输完毕，关闭套接字。



TCP通信模型

4．头文件

Berkeley 套接字接口的定义在几个头文件中。这些文件的名字和内容与具体的实现之间有些许的不同。大体上包括：

<sys/socket.h>：核心 BSD 套接字核心函数和数据结构。AF_INET、AF_INET6 地址集和它们相应的协议集 PF_INET、PF_INET6. 广泛用于 Internet，这些包括了 IP 地址和 TCP、UDP 端口号。

<netinet/in.h>：AF_INET 和 AF_INET6 地址家族和他们对应的协议家族 PF_INET 和 PF_INET6。在互联网编程中广泛使用，包括 IP 地址以及 TCP 和 UDP 端口号。

<sys/un.h>：PF_UNIX/PF_LOCAL 地址集。用于运行在一台计算机上的程序间的本地通信，不用于网络通讯。

<arpa/inet.h>：处理数值型 IP 地址的函数。

<netdb.h>：将协议名和主机名翻译为数值地址的函数。搜索本地数据以及 DNS。

5．套接字 API 函数

这个列表是一个 Berkeley 套接字 API 库提供的函数或者方法的概要：

（1）socket() 创建一个新的确定类型的套接字，类型用一个整型数值标识（文件描述符），并为它分配系统资源。

（2）bind() 一般用于服务器端，将一个套接字与一个套接字地址结构相关联，比如，一个指定的本地端口和 IP 地址。

（3）listen() 用于服务器端，使一个绑定的 TCP 套接字进入监听状态。

（4）connect() 用于客户端，为一个套接字分配一个自由的本地端口号。如果是 TCP 套接字的话，它会试图获得一个新的 TCP 连接。

（5）accept() 用于服务器端。它接受一个从远端客户端发出的创建一个新的 TCP 连接的接入请求，创建一个新的套接字，与该连接相应的套接字地址相关联。

（6）send()和 recv(),或者 write()和 read(),或者 recvfrom()和 sendto(), 用于往/从远程套接字发送和接受数据。

（7）close() 用于系统释放分配给一个套接字的资源。如果是 TCP，连接会被中断。

（8）inet_addr()用于将一个网络地址字符串转换成一个 32 位二进制网络 IP 地址。

（9）htons()用于将一个 int 端口号转换成一个 16 位二进制网络字节序。

四、实验内容

1. 通过实验指导书或者网络资源，学习 TCP 套接字原理及应用方法。

2. 请补充以下 server.c 和 client.c 中代码。本实验中实验箱和 Ubuntu，把其中一个当作服务器，一个当作客户端，执行编译生成的可执行程序，进行通信。观察通信结果。

（1）server.c 中部分代码如下，可把文档放大后再查看代码：

```
#include <sys/types.h>
#include <sys/socket.h>                    // 包含套接字函数库
#include <stdio.h>
#include <netinet/in.h>                    // 包含 AF_INET 相关结构
```

```
#include <arpa/inet.h>                    // 包含 AF_INET 相关操作的函数
#include <unistd.h>
#include <string.h>                       //bzero()

#define MAXDATASIZE 50

int main()
{
    int server_sockfd;                    //服务器端 socket 描述符变量,用以监听客户端的 TCP 连接
    int new_sockfd;                       //当监听到并接受一个 TCP 连接时，accept()返回一个新的 socket，用这个 socket 接受数据
    struct sockaddr_in server_addr; //服务器端 sockaddr_in 的数据结构
    struct sockaddr_in client_addr; //用于 accept()存放客户端的 socket 数据结构
    int client_addr_len;                  //用于返回 struct sockaddr_in 的长度，作为 accept()中的一个参数
    char buf[MAXDATASIZE];                //创建一个 buf 来存放接受的数据
    int numbytes;                         //用以返回接受到的字节数

    server_sockfd = socket(AF_INET , SOCK_STREAM , 0); //创建一个 TCP 套接字,其中协议族为 AF_INET,套接字类型为 SOCK_STREAM,第三个参数为 0
    server_addr.sin_family = AF_INET; //server_addr.sin_family 为 AF_INET
    server_addr.sin_addr.s_addr =inet_addr("10.10.58.213");    //server_addr.sin_addr.s_addr 为服务器端 IP 地址，需经 inet_addr()转化为网络 IP 字节序
    server_addr.sin_port =htons(1025); //server_addr.sin_port 为一个大于 1024 的端口号，需经 htons()转化为网络字节序
    bind(server_sockfd , (struct sockaddr *)&server_addr ,  sizeof(struct sockaddr)); // 将 server_sockfd 绑定到 server_addr 指定的 IP 和端口上
    listen(server_sockfd, 5);             //为 server_sockfd 创建包含 5 个 TCP 连接的监听队列，将其设置为监听套接字
    while (1) {
        client_addr_len = sizeof(struct sockaddr_in); //计算 sockaddr_in 的长度
        new_sockfd= accept(server_sockfd , (struct sockaddr *)&client_addr, &numbytes);//等待接受客户端发来的 TCP 连接，若连接成功，返回新的 socket
        numbytes = recv(new_sockfd, buf, 20, 0); //将客户端发来的数据存放在 buf 中，返回接受到的字节数
        if (numbytes)
        {
            printf("%s\n",buf); //输出 buf 中数据
        }
        sleep(3);
        send(new_sockfd, "I am server", 20, 0);    //向客户端发送"I am server"
        close(new_sockfd);
        if('8'==getchar())
```

```
        {
            break;
        }
    }
    close(server_sockfd);
    return 0;
}
```

（2）client.c 中部分代码如下，可把文档放大后再查看代码：

```
#include <sys/types.h>
#include <sys/socket.h>                  // 包含套接字函数库
#include <stdio.h>
#include <netinet/in.h>                  // 包含 AF_INET 相关结构
#include <arpa/inet.h>                   // 包含 AF_INET 相关操作的函数
#include <unistd.h>
#include <string.h>                      //bzero()

#define MAXDATASIZE 50

int main(int argc, char *argv[])
{
    int sockfd,result; //客户端 socket 描述符变量
    struct sockaddr_in address_addr;//与服务器端 sockaddr_in 的数据结构一致
    char buf[MAXDATASIZE]; //用以接受服务器端发送过来的数据
    int numbytes; //用以返回从服务端发送过来的数据字节数
    sockfd = socket(AF_INET , SOCK_STREAM , 0); //创建一个 TCP 套接字,其中协议族为 AF_INET,套接字类型为 SOCK_STREAM,第三个参数为 0
    address_addr.sin_family = AF_INET;//address_addr_addr.sin_family 为 AF_INET
    address_addr.sin_addr.s_addr =inet_addr("10.10.58.213"); //address_addr_addr.sin_addr.s_addr 为服务器端 IP 地址，需经 inet_addr()转化为网络 IP 字节序
    address_addr.sin_port = htons(1025);//address_addr_addr.sin_port 为一个大于 1024 的端口号，需经 htons()转化为网络字节序
    result = connect(sockfd, (struct sockaddr *)&address_addr,sizeof(struct sockaddr));// 与 address_addr_addr 指定的服务器建立连接
    send(sockfd, "This is client", 20, 0); //向服务器端发送"this is client"
    sleep(3);
    numbytes = recv(sockfd, buf ,20 , 0); //接受服务器端发送过来的数据,保存在 buf 中
    if (numbytes)
    {
        printf("%s\n",buf); //输出 buf 里面的数据
    }
```

```
    close(sockfd);
    return 0;
}
```

五、实验结果及分析

1．简述 TCP 协议与 UDP 协议原理

TCP/IP 协议是指传输控制协议/因特网互联协议，又叫网络通讯协议，是由网络层的 IP 协议和传输层的 TCP 协议组成的。TCP/IP 定义了电子设备（比如计算机）如何连入因特网，以及数据如何在它们之间传输的标准。从协议分层模型方面来讲，TCP/IP 由四个层次组成：网络接口层、网络层、传输层、应用层。 每一层都呼叫它的下一层所提供的网络来完成自己的需求。

UDP，即用户数据报协议。作为运输层协议，UDP 使用端口号来完成进程到进程之间的通信，UDP 在运输层提供非常有限的流控制机制，在收到分组时没有流控制也没有确认。但是，UDP 提供了某种程度的差错控制。如果 UDP 检测出在收到的分组有一个差错，它就悄悄的丢弃这个分组。UDP 不负责为进程提供连接机制，它只从进程接收数据单元，并将他们不可靠的交付给接收端。UDP 提供的是无连接的、不可靠的运输服务。

2．描述 TCP 服务器端套接字通信过程。

（1）调用 socket()创建套接字，然后初始化 struct sockaddr_in 结构体。

（2）调用 bind 函数()为套接字绑定一个 IP 地址和一个端口号。

（3）调用 listen()函数使套接字成为监听套接字，侦听指定的端口。

（4）调用 accept()函数，使服务器处于阻塞状态，等待接受客户端连接请求。一旦建立连接，将产生新的套接字，此时就有两个套接字了，原来的那个套接字还在监听等待指定的端口，而新产生的套接字则准备发送或接受数据。

（5）利用 send/sendto 和 recv/recvfrom 进行数据传输。当然也可以调用 write 或 read.

（6）数据传输完毕，关闭套接字。

3．描述 TCP 客户端端套接字通信过程。

（1）调用 socket()创建套接字，然后初始化 struct sockaddr_in 结构体，注意服务器端和客户端的 struct sockaddr_in 结构体应该一致。

（2）调用 connection()函数与服务器建立连接。

（3）利用 send/sendto 和 recv/recvfrom 进行数据传输. 当然也可以调用 write 或 read.

（4）数据传输完毕，关闭套接字。

4．自己尝试编写 UDP 套接字通信程序，并进行测试。