Linux网络编程基础

这里对计算机网络方面的内容就只作简单的介绍，不牵涉太多学术化的概念。

参考书目：《Linux网络编程》、*《Linux/Unix系统编程手册》*、《计算机网络:自顶而下的方法》

前置知识

OSI七层网络模型、TCP/IP协议栈

socket是一种IPC方法，允许同一主机或使用网络连接起来的不同主机上的应用程序之间交换数据。传输层有TCP协议和UDP协议，这个网上有很多资料。

TCP协议 编程模型：

服务端	客户端
创建socket	创建socket
确定服务器地址簇	获取服务器地址协议簇
绑定	
监听	
接收连接	连接服务器
通信	通信
断开连接	断开连接

UDP协议 编程模型

"服务端"	"客户端"
创建socket	创建socket
确定服务端地址协议簇	获取客户端地址协议簇
绑定	
通信	通信

创建socket采用socket函数

```
NAME
       socket - create an endpoint for communication

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);

DESCRIPTION
       socket()  creates  an  endpoint for communication and returns a file descriptor that refers to that endpoint.  The file
       descriptor returned by a successful call will be the  lowest-numbered  file  descriptor  not  currently  open  for  the
       process.
```

```
   The  domain argument specifies a communication domain; this selects the protocol family which will be used for communi‐
   cation.  These families are defined in <sys/socket.h>.  The currently understood formats include:

   Name                Purpose                          Man page
   AF_UNIX, AF_LOCAL   Local communication              unix(7)
   AF_INET             IPv4 Internet protocols          ip(7)
   AF_INET6            IPv6 Internet protocols          ipv6(7)
   AF_IPX              IPX - Novell protocols
   AF_NETLINK          Kernel user interface device     netlink(7)
   AF_X25              ITU-T X.25 / ISO-8208 protocol   x25(7)
   AF_AX25             Amateur radio AX.25 protocol
   AF_ATMPVC           Access to raw ATM PVCs
   AF_APPLETALK        AppleTalk                        ddp(7)
   AF_PACKET           Low level packet interface       packet(7)
   AF_ALG              Interface to kernel crypto API
```

```
  The socket has the indicated type, which specifies the communication semantics.  Currently defined types are:

  SOCK_STREAM     Provides sequenced, reliable, two-way, connection-based byte streams.  An out-of-band data transmission
                  mechanism may be supported.

  SOCK_DGRAM      Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

  SOCK_SEQPACKET  Provides  a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed
                  maximum length; a consumer is required to read an entire packet with each input system call.

  SOCK_RAW        Provides raw network protocol access.

  SOCK_RDM        Provides a reliable datagram layer that does not guarantee ordering.

  SOCK_PACKET     Obsolete and should not be used in new programs; see packet(7).
```

```
   The  protocol  specifies  a  particular protocol to be used with the socket.  Normally only a single protocol exists to
   support a particular socket type within a given protocol family, in which case protocol can be specified  as  0.   How‐
   ever,  it is possible that many protocols may exist, in which case a particular protocol must be specified in this man‐
   ner.  The protocol number to use is specific to the “communication domain” in which communication is to take place; see
   protocols(5).  See getprotoent(3) on how to map protocol name strings to protocol numbers.
```

TCP网络编程

将socket函数的第一个参数domain(通信域)指定为AF_INET，第二个参数是socket类型，这里传入SOCK_STREAM，流式套接字。因为只使用一种协议，这里将第三个参数设置为0

创建完套接字后，要对sockaddr_in结构体进行初始化，struct sockaddr是通用地址结构。

使用bind函数进行绑定，将一个socket绑定到一个地址上

```
NAME
       bind - bind a name to a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int bind(int sockfd, const struct sockaddr *addr,
                socklen_t addrlen);

DESCRIPTION
       When  a socket is created with socket(2), it exists in a name space (address family) but has no address assigned to it.
       bind() assigns the address specified by addr to the socket referred to by the file descriptor sockfd.   addrlen  speci‐
       fies the size, in bytes, of the address structure pointed to by addr.  Traditionally, this operation is called “assign‐
       ing a name to a socket”.
```

使用listen函数进行监听

```
NAME
       listen - listen for connections on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int listen(int sockfd, int backlog);

DESCRIPTION
       listen()  marks  the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to accept
       incoming connection requests using accept(2).

       The sockfd argument is a file descriptor that refers to a socket of type SOCK_STREAM or SOCK_SEQPACKET.

       The backlog argument defines the maximum length to which the queue of pending connections for sockfd may  grow.   If  a
       connection  request  arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED
       or, if the underlying protocol supports retransmission, the request may be ignored so that a later reattempt at connec‐
       tion succeeds.

RETURN VALUE
       On success, zero is returned.  On error, -1 is returned, and errno is set appropriately.
```

使用accept函数进行接收套接字

```
NAME
       accept, accept4 - accept a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

       #define _GNU_SOURCE             /* See feature_test_macros(7) */
       #include <sys/socket.h>

       int accept4(int sockfd, struct sockaddr *addr,
                   socklen_t *addrlen, int flags);

DESCRIPTION
       The  accept()  system  call  is used with connection-based socket types (SOCK_STREAM, SOCK_SEQPACKET).  It extracts the
       first connection request on the queue of pending connections for the listening socket, sockfd, creates a new  connected
       socket,  and  returns a new file descriptor referring to that socket.  The newly created socket is not in the listening
       state.  The original socket sockfd is unaffected by this call.
```

使用recv和send函数可以收发数据

```
NAME
       recv, recvfrom, recvmsg - receive a message from a socket

SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       ssize_t recv(int sockfd, void *buf, size_t len, int flags);

       ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                        struct sockaddr *src_addr, socklen_t *addrlen);

       ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);

DESCRIPTION
       The  recv(),  recvfrom(),  and recvmsg() calls are used to receive messages from a socket.  They may be used to receive
       data on both connectionless and connection-oriented sockets.  This page first describes common features  of  all  three
       system calls, and then describes the differences between the calls.
```

```
NAME
       send, sendto, sendmsg - send a message on a socket

SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       ssize_t send(int sockfd, const void *buf, size_t len, int flags);

       ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
                      const struct sockaddr *dest_addr, socklen_t addrlen);

       ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);

DESCRIPTION
       The system calls send(), sendto(), and sendmsg() are used to transmit a message to another socket.

       The  send()  call  may  be used only when the socket is in a connected state (so that the intended recipient is known).
       The only difference between send() and write(2) is the presence of flags.  With a zero flags argument, send() is equiv‐
       alent to write(2).  Also, the following call
```

服务端代码如下

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <stdbool.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>

int serv_sock,clnt_sock;

void hand() {
    // 断开连接
    close(clnt_sock);
    close(serv_sock);
    printf("bye\n");
    exit(0);
}

int main(int argc,char* argv[])
{
    if (argc != 3) {
        printf("input error\n");
        exit(-1);
    }

    signal(2,hand);

    // 创建socket
    serv_sock = socket(AF_INET,SOCK_STREAM,0);
    if (serv_sock == -1) {
        printf("socket error:%m\n");
        exit(-1);
    }
    printf("socket success\n");

    // 创建套接字地址协议簇
    struct sockaddr_in serv_addr = {0};
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]); // IP地址 字符串->点分
    serv_addr.sin_port = htons(atoi(argv[2])); // 转字节序

    // 绑定
    int ret = bind(serv_sock,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    if (ret == -1) {
        printf("bind error:%m\n");
        close(serv_sock);
        exit(-2);
    }
    printf("bind success\n");

    // 监听
    ret = listen(serv_sock,10);
    if (-1 == ret) {
        printf("listen error:%m\n");
        close(serv_sock);
        exit(-2);
    }
    printf("listen success\n");

    struct sockaddr_in clnt_addr = {0};
    int len = sizeof(clnt_addr);
    // 接收连接
    clnt_sock = accept(serv_sock,(struct sockaddr*)&clnt_addr,&len);
    if (clnt_sock == -1) {
        printf("accept error:%m");
        close(serv_sock);
        exit(-2);
    }
    printf("accept success\n");
    printf("%s is connecting\n",inet_ntoa(clnt_addr.sin_addr));

    // 通信
    char buff[256] = {0};
    while (true) {
        ret = recv(clnt_sock,buff,256,0);
        if (ret > 0) {
            buff[ret] = 0;
            printf(">> %s\n",buff);
        }
    }
    return 0;
}
```

要注意的是accept函数是一个阻塞函数，没有连接时，就会卡在那等待。

客户端调用connect连接服务器

```
NAME
       connect - initiate a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int connect(int sockfd, const struct sockaddr *addr,
                   socklen_t addrlen);

DESCRIPTION
       The  connect()  system  call  connects the socket referred to by the file descriptor sockfd to the address specified by
       addr.  The addrlen argument specifies the size of addr.  The format of the address in addr is determined by the address
       space of the socket sockfd; see socket(2) for further details.
```

客户端代码如下

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <stdbool.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>

int clnt_sock;

int main(int argc,char* argv[])
{
    if (argc != 3) {
        printf("input error\n");
        exit(-1);
    }

    clnt_sock = socket(AF_INET,SOCK_STREAM,0);
    if (clnt_sock == -1) {
        printf("socket error:%m\n");
        exit(-1);
    }
    printf("socket success\n");
    struct sockaddr_in serv_addr;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));

    // 连接服务器
    int ret = connect(clnt_sock,(struct
sockaddr*)&serv_addr,sizeof(serv_addr));
    if (ret == -1) {
        printf("connect error:%m\n");
        exit(-1);
    }
    printf("connect success\n");

    char buff[256] = {0};
    while (true) {
        printf("<< ");
        scanf("%s",buff);
        ret = send(clnt_sock,buff,strlen(buff),0);
    }
    return 0;
}
```

先运行服务端，再运行客户端，运行结果如下：

```
hwx@N3ptune:/data/home/hwx/cprogram/testc/linux-network/tcp$ ./tcp_server 127.0.0.1 9527
socket success
bind success
listen success
accept success
127.0.0.1 is connecting
>> hello
>> bye!
```

```
hwx@N3ptune:/data/home/hwx/cprogram/testc/linux-network/tcp$ ./tcp_client 127.0.0.1 9527
socket success
connect success
<< hello
<< bye!
```

将上述过程简单描述就是

下面简单实现一个文件传输

接收端是服务器，发送端是客户端。流程如下：

服务端	客户端
等待客户端发送	发送文件名
接收文件名并创建文件	获取并发送文件大小
接收文件大小	打开文件准备读取发送
循环接收并写入数据	循环读取并发送
接收数据完成关闭文件和连接	发送完成并关闭文件

如下是接收端的代码

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdbool.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>

#define ADDR "127.0.0.1"
#define PORT 9527

int serv_sock,clnt_sock;

int main(void)
{
    serv_sock = socket(AF_INET,SOCK_STREAM,0);
    if (serv_sock == -1) {
        printf("socket error:%m\n");
        exit(-1);
    }
    printf("socket success\n");

    struct sockaddr_in serv_addr = {0};
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(ADDR);
    serv_addr.sin_port = htons(PORT);

    int ret = bind(serv_sock,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    if (ret == -1) {
        printf("bind error:%m\n");
        exit(-1);
    }
    printf("bind success\n");

    ret = listen(serv_sock,10);
    if (ret == -1) {
        printf("listen error:%m\n");
        exit(-1);
    }
    printf("listen success\n");

    struct sockaddr_in clnt_addr = {0};
    int len = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock,(struct sockaddr*)&clnt_addr,&len);
    if (clnt_sock == -1) {
        printf("accept error:%m\n");
        exit(-1);
    }
    printf("accept success\n");
    printf("%s is connecting\n",inet_ntoa(clnt_addr.sin_addr));

    char buff[1024] = {0}; // 缓冲区 存放文件内容
    char name[256] = {0};  // 文件名
    int size = 0;
    // 先接收文件名和文件大小
    ret = recv(clnt_sock,name,255,0);
    if (ret > 0) {
        name[ret] = 0;
        printf("File name: %s\n",name);
    }
    ret = recv(clnt_sock,(char*)&size,4,0);
    if (ret == 4) {
        printf("File size: %d\n",size);
    }

    int count = 0; // 已经接收的大小
    int fd = open("./recv.txt",O_CREAT|O_WRONLY,0666); // 打开文件
    while (true) {
        ret = recv(clnt_sock,buff,1024,0);
        if (ret > 0) {
            write(fd,buff,ret);
            count += ret;
            printf("recv size: %d\n",count);
            if (count >= size)
                break;
        }
        else
            break;
    }

    close(fd);
    sleep(1);
    printf("receive ok\n");
    close(clnt_sock);
    close(serv_sock);
    return 0;
}
```

如下是发送端

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdbool.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>

#define ADDR "127.0.0.1"
#define PORT 9527

int clnt_sock;

int main(int argc,char* argv[])
{
    clnt_sock = socket(AF_INET,SOCK_STREAM,0);
    if (clnt_sock == -1) {
        printf("socket error:%m\n");
        exit(-1);
    }
    printf("socket success\n");

    struct sockaddr_in serv_addr = {0};
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(ADDR);
    serv_addr.sin_port = htons(PORT);

    int ret = connect(clnt_sock,(struct
sockaddr*)&serv_addr,sizeof(serv_addr));
    if (ret == -1) {
        printf("connect error:%m\n");
        exit(-1);
    }
    printf("connect success\n");

    char name[256] = {0};
    printf("input the file name: ");
    scanf("%s",name);
    // 打开文件
    int fd = open(name,O_RDONLY);
    if (fd == -1) {
        printf("open error:%m\n");
        exit(-1);
    }
    printf("open %s successfully\n",name);

    struct stat st = {0};
    stat(name,&st);
    printf("File size: %d\n",st.st_size);

    send(clnt_sock,name,sizeof(name),0);
    send(clnt_sock,(char*)&st.st_size,4,0);

    char buff[1024] = {0};
    while(true) {
        ret = read(fd,buff,1024);
        if (ret > 0) {
            send(clnt_sock,buff,ret,0);
        }
        else
            break;
    }
    printf("send ok\n");
    sleep(1);
    close(clnt_sock);

    return 0;
}
```

```
Hello Linux
```

这里将上级目录中的一个test.txt文本文件发送，服务端保存在本地为recv.txt

运行结果

```
hwx@N3ptune:/data/home/hwx/cprogram/testc/linux-network/tcp$ ./file_recv
socket success
bind success
listen success
accept success
127.0.0.1 is connecting
File name: ../test.txt
File size: 11
recv size: 11
receive ok
```

```
hwx@N3ptune:/data/home/hwx/cprogram/testc/linux-network/tcp$ ./file_send
socket success
connect success
input the file name: ../test.txt
open ../test.txt successfully
File size:11
send ok
```

已经成功保存

下面介绍UDP

UDP网络编程

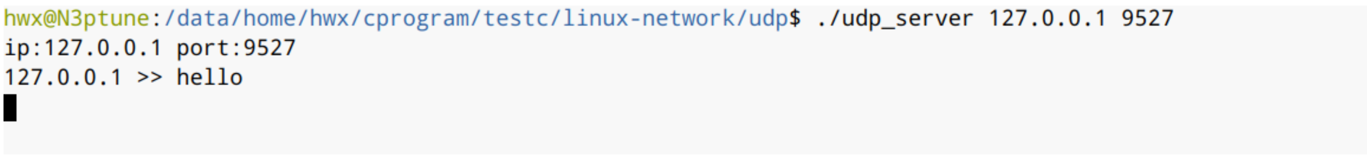
在socket函数中，协议类型换成SCOK_DGRAM

如下是服务端代码，只要做套接字和绑定操作

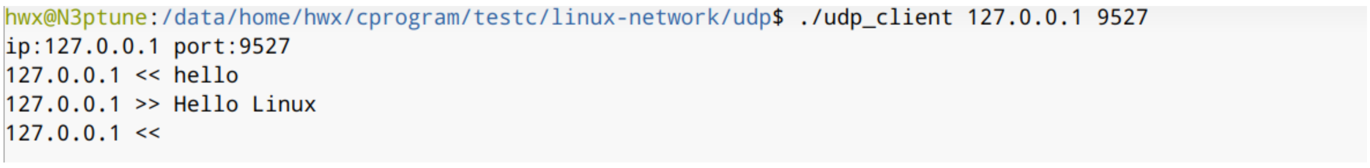
```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <stdbool.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>

int serv_sock;

void hand()
{
    close(serv_sock);
    printf("bye\n");
    exit(0);
}

int main(int argc,char* argv[])
{
    if (argc != 3) {
        printf("input error\n");
        exit(0);
    }
    signal(2,hand);
    printf("ip:%s port:%s\n",argv[1],argv[2]);

    serv_sock = socket(AF_INET,SOCK_DGRAM,0);
    if (-1 == serv_sock) {
        printf("socket error:%m\n");
        exit(-1);
    }

    struct sockaddr_in serv_addr = {0};
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));
    int ret = bind(serv_sock,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
    if (ret == -1) {
        printf("bind error:%m\n");
        exit(-2);
    }

    char buff[256] = {0};
    struct sockaddr_in clnt_addr = {0};
    int len = sizeof(clnt_addr);
    while(true) {
        ret = recvfrom(serv_sock,buff,255,0,
            (struct sockaddr*)&clnt_addr,&len);
        if (ret > 0) {
            buff[ret] = 0;
            printf("%s >> %s\n",inet_ntoa(clnt_addr.sin_addr),buff);
        }
        sendto(serv_sock,"Hello Linux",sizeof("Hello Linux"),
            0,(struct sockaddr*)&clnt_addr,sizeof(clnt_addr));
    }

    return 0;
}
```

如下是客户端代码

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <stdbool.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>

int clnt_sock;

void hand()
{
    close(clnt_sock);
    printf("bye\n");
    exit(0);
}

int main(int argc,char* argv[])
{
    if (argc != 3) {
        printf("input error\n");
        exit(0);
    }
    signal(2,hand);
    clnt_sock = socket(AF_INET,SOCK_DGRAM,0);
    if (clnt_sock == -1) {
        printf("socket error:%m\n");
        exit(-1);
    }

    struct sockaddr_in serv_addr;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));
    printf("ip:%s port:%s\n",argv[1],argv[2]);

    int len = sizeof(serv_addr);
    char buff[256] = {0};
    char temp[256] = {0};
    while(true) {
        printf("%s << ",argv[1]);
        scanf("%s",buff);
        sendto(clnt_sock,buff,sizeof(buff),0,
            (struct sockaddr*)&serv_addr,sizeof(serv_addr));
        int ret = recvfrom(clnt_sock,temp,255,0,
                (struct sockaddr*)&serv_addr,&len);

        if (ret > 0) {
            temp[ret] = 0;
            printf("%s >> %s\n",argv[1],temp);
        }
    }

    return 0;
}
```



```
hwx@N3ptune:/data/home/hwx/cprogram/testc/linux-network/udp$ ./udp_server 127.0.0.1 9527
ip:127.0.0.1 port:9527
127.0.0.1 >> hello
```



```
hwx@N3ptune:/data/home/hwx/cprogram/testc/linux-network/udp$ ./udp_client 127.0.0.1 9527
ip:127.0.0.1 port:9527
127.0.0.1 << hello
127.0.0.1 >> Hello Linux
127.0.0.1 <<
```