Design and Research of Future Network (IPV9) API

Xu Yinqiu
Shanghai Decimal Network Information Technology Co. Ltd.
E-mail: 8918616209@126.com

Xie Jianping
Shanghai Decimal Network Information Technology Co. Ltd.
E-mail: 13386036170@189.cn

***Abstract*—Socket is a way of process communication, that is used it to invoke some API function to realize the distribution network libraries in different host of data exchange between the relevant process. According to the TCP/IP protocol assigned to the network address of the local host, to communicate between the two processes, the host must know the other's location first, that is, the IP of the other host. At the same time, to get the port number, it is used to identify the local communication process; a local process in communication will occupy a port number, different process port number is different, so it must be assigned a port number that is not used before communication. A complete inter-network process consists of two processes and should use the same high-level protocol. IPV9 is the most important part of future network. This paper introduces the interface function and socket of IPV9, which lays a foundation for further network application programming.**

Keywords -IPV9; Interface; Socket; API

I. INTERFACE AND SOCKET

The transport layer implements end-to-end communication, so there are two terminals for each transport layer connection. What is the terminal of the transport layer connection? It is neither the host, nor the host's IP address, and not the application process, not the transport layer protocol port. The terminal to which the transport layer connects is called a socket. According to the definition of RFC793, the port number is spliced to the IP address to form a socket. A socket is actually a communication endpoint, an interface between an application and a network protocol. Each socket has a socket number, including the IP address of the host and a 16-bit host port number, such as (host IP address: port number).

In short, Socket is equals to (IP address: port number), which is represented by a decimal IP address followed by a port number, separated by a colon or comma. Each transport layer connection is uniquely identified by two terminals (that is, two sockets) at each end of the communication. For example, if the IPv4 address is 118.38.18.1 and the port number is 23, the resulting socket is (118.38.18.1:23), If the IPV9 address is 86[128[5]118.38.18.1 and the port number is 23, the resulting socket is (86[128[5] 18.38.18.1:23).

A socket can be thought of as a terminal in the communication connection between two network applications. During communication, a piece of information to be transmitted by one of the network applications is written into the Socket of its host, which sends the piece of information to the Socket of another host through the transmission medium of the network interface, so that the piece of information can be transmitted to other programs. Therefore, the data transfer between the two applications is done through the socket.

During network application design, IPv4 can be realized through the programming interface of TCP/IP provided by the system, since the core content of TCP/IP is encapsulated in the operating system.

All clients and servers of TCP-based socket programming begin with calling a socket, which returns a socket descriptor. The client then calls the connect function, while the server calls the bind (), listen (), and accept () functions. The socket is usually closed by using the standard close function, but it can be also used the shutdown function to close the socket. The Socket interaction flow is shown in figure 1.

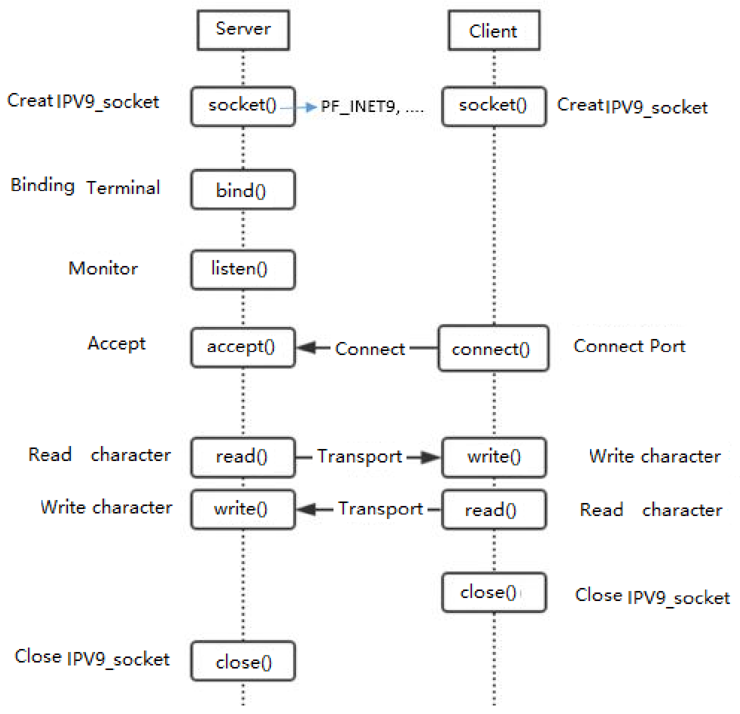

Figure 1. The Socket interaction flow

II. IPV9 SOCKET

In the Linux environment of IPv9, the core contents of TCP 9/IP9 are encapsulated in the operating system kernel. In order to support user development of application-oriented communication programs, most systems provide a set of application programming interfaces (API) based on TCP 9 or UDP 9, which are usually presented in the form of a set of functions, also known as sockets. These sockets are described below. This document is the IPv9 protocol experimental application development instructions, non-industry standard documents.

A. Socket

A socket is an abstraction layer through which applications can send or receive data and open, read, and close it as if it were a file.Sockets allow applications to plug I/O into the network and communicate with other applications in the network. This version of network sockets supports a combination of IPv4, IPv6, and IPv9 addresses and ports.

1) Head file

#include<sys/types.h>

#include<sys/socket.h>

2) Prototype

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

3) Description

Socket: Creates and returns a communication sleeve interface handle.

The parameter domain describes the communication domain, that is, the select communication protocol family. These communication protocol families are defined in the header file

<sys/socket.h>. Currently supported protocol families are as follows:

- PF_UNIX,PF_LOCAL(Local communication protocol)
- PF_INETIPv4 (Protocol)
- PF_INET6(IPv6 Protocol)
- PF_INET9(IPv9 Protocol)
- PF_IPXNovell (Protocol)
- PF_NETLINK(Core user interface device)
- PF_X25ITU-T X.25 (Protocol)
- PF_AX25AX.25 (Protocol)
- PF_ATMPVC(Access to the original ATM PVCs)
- PF_APPLETALKAppletalk
- PF_PACKET(Low-level envelope interface)

The parameter type is used to describe the communication semantics. Currently defined types are as follows:

- SOCK_STREAM

It provides sequential, reliable, duplex, connection-based byte streams that can also support out-of-band data transfer.

- SOCK_DGRAM

It supports datagram (connectionless, unreliable messages of fixed maximum length).

- SOCK_SEQPACKET

It provides a sequential, reliable, duplex, connection-based data path for datagram of fixed maximum length.

- SOCK_RAW

It provides original network protocol access.

- SOCK_RDM

It provides a reliable datagram layer that does not guarantee order.

Some sets of interface types are not implemented on all protocol families, such as the SOCK SEQPACKET is not implemented in the AF_INET protocol family

The parameter protocol describes a special protocol for the socket interface. There is usually only one simple protocol that can support a particular set of interface types that contain a given family of protocols. Of course, sometimes when multiple protocols exist that must be specified with this parameter.

4) Returned value

-1 is returned value when an error occurs, and errno represents the error type value. Otherwise, the socket interface handle value is returned。

B. Bind ()

Bind () is a local address to a set of interfaces function. This function is suitable for unconnected datagram or stream class interfaces and is used before connect () or listen () calls. When a socket () is created, it exists in a namespace (address family) but is not named. The bind () function establishes a local binding (host address/port number) for the socket interface by assigning a local name to an unnamed socket interface.

1) Head file

#include<sys/types.h>

#include<sys/socket.h>

2) Prototype

Bind(intsockfd, structsockaddr *my_addr, socklen_taddrlen);

3) Description

Bind() provides the local address my_addr for the socket interface handle, the length of my_addr is the parameter addrlen, which is called set interface name assignment.

In general, a socket interface of type SOCK_STREAM must call bind() to assign a local address in order to connect and receive.

The structure of the assignment is also different for different protocol families. Such as for AF_INET is sockaddr_in and AF_INET9 is sockaddr_in9.

4) Returned value

Return 0 on success. The error returns -1, and errno represents the error type value.

C. Connect ()

Connect () is used to establish a connection to the specified socket。

1) Head file

#include <sys/types.h>

#include<sys/socket.h>

2) Prototype

Connect (intsockfd, conststructsockaddr *serv_addr, socklen_taddrlen);

3) Description

The handle sockfd must point to a socket interface. If the type of the socket interface is SOCK_DGRAM, the address represented by the parameter serv_addr is the default destination address of the datagram and the source address when the datagram is received. If the socket interface is of type SOCK_STREAM or SOCK_SEQPACKET, the call attempts to establish a connection to another socket interface. The other interface is described by the serv_addr parameter, which is the address of interface communication spaces, each of which interprets the serv_addr parameter.

Typically, connection-based protocols only successfully connect once; connectionless interfaces may connect multiple times to change sessions. A connectionless interface may also connect to an address whose family of protocols is AF_UNSPEC to cancel the session.

4) Returned value

Return 0 on success. The error returns -1, and errno represents the error type value.

D. Listen ()

It is used to create a socket interface and listen for the requested connection.

1) Head file

#include <sys/types.h>

#include<sys/socket.h>

2) Prototype

int listen(int s, int backlog);

3) Description

To confirm the connection, the socket is called to create a socket interface, and the listen () describes the willing to confirm the connection and the length limit of the connection queue before calling accept to confirm the connection. The listen () call only works on the socket interfaces of types SOCK_STREAM and SOCK_SEQPACKET.

The parameter backlog defines the maximum length of the unconnected queue.

4) Returned value

Return 0 on success. The error returns -1, and errno represents the error type value.

E. Accept ()

It is used to create a socket interface and monitoring for the requested connection.

1) Head file

#include <sys/types.h>

#include<sys/socket.h>

2) Prototype

int accept(int s, structsockaddr *addr, socklen_t *addrlen);

3) Description

The accept function can be used based on the socket interface type of the connection (SOCK_STREAM, SOCK_SEQPACKET, and SOCK_RDM). It selects the first connection request in the unconnected queue, creates a new connected socket interface similar to the parameter s, and then assigns a handle to that socket interface and returns. The newly created socket interface is no longer in the listening state, and the source socket interface s is not affected by the call.

4) Returned value

The error returns -1, and errno represents the error type value. Successfully returns a non-negative integer, representing the handle to the socket interface.

F. Select ()

It is used for monitoring three socket interfaces.

1) Head file

#include <sys/time.h>

#include <sys/types.h>

#include <unistd.h>

2) Prototype

int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,

structtimeval*timeout);

FD_CLR(intfd, fd_set *set);

FD_ISSET(intfd, fd_set *set);

FD_SET(intfd, fd_set *set);

FD_ZERO(fd_set *set);

3) Description

Select () allows to monitoringthe three socket interface at the same time: readfds, writefds and exceptfds.

The socket interface in the Readfds will be listened for acceptable characters; the socket interface in writefds will be monitored to see if data can be sent immediately. The socket interface in exceptfds will be monitored for exceptions.

Four macros are defined to manipulate the set of socket interfaces: FD_ZERO will empty a set; FD_SET and FD_CLR add or remove a handle from the set. FD_ISSET is used to test whether a handle is in the set.

The parameter n should be equal to the value of the highest file descriptor plus 1.

The timeout parameter defines the maximum interval for the select call to block until it returns. It can be zero, so that select returns directly. If the timeout structure pointer is NULL, select will block the indeterminate time.

4) Returned value

On success, return the socket interface handle contained in the socket interface set, returns 0 if no change occurs after the maximum interval. -1 is returned on error, and errno represents the error type value

G. Recv (), Recvfrom(), Recvmsg()

1) Head file

#include <sys/types.h>

#include<sys/socket.h>

2) Prototype

intrecv(int s, void *buf, size_tlen, int flags);

Intrecvfrom(int s, void *buf, size_tlen, int flags, structsockaddr *from,

socklen_t *fromlen);

Intrecvmsg(int s, structmsghdr *msg, int flags);

3) Description

The Recvfrom() and recvmsg() calls are used to receive information from a socket interface, regardless of whether the socket interface is connection-oriented. If the from () parameter is not NULL, then the socket interfaces is not connection-oriented and the source address of the message is assigned to it. The fromlen parameter starts with the data buffer size of the parameter from and returns the buffer size of the actual storage address in the parameter from.

A Recv () call is usually used in a connected socket interface, this is equivalent to the case where the parameter from is NULL when recvfrom is called.

If the data message is successfully received, the return value is the length of the data message. If the length of the data message exceeds the length of the data buffer, the excess is discarded, depending on the type of socket interface used to receive the message.

If the socket interface does not receive the information, it will always wait for the information unless the socket interface is non-blocking. When the socket interface is non-blocking, the return value is -1 and the errno value is EAGAIN.

The Recvmsg call the MSGHDR structure, defined in the header file <sys/socket.h>.

4) *Returned value*

The length of the received data is returned on success, -1 is returned on error, and errno represents the value of the error type.

H. Send(), Sendto(), Sendmsg()

1) *Head file*

#include <sys/types.h>

#include<sys/socket.h>

2) *(2) Prototype*

Intsend(int s, const void *msg, size_tlen, int flags);

Intsendto(int s, const void *msg, size_tlen, int flags, conststructsockaddr *to,

socklen_ttolen);

Intsendmsg(int s, conststructmsghdr *msg, int flags);

3) *Description*

The Send (), sendto, and sendmsg calls are used to transfer information to other interfaces. The Send () call applies only to the connection-oriented socket interface, while the sendto and sendmsg calls apply to all situations.

The destination address is set by the parameter to, its length is the parameter tolen, and the length of the message is represented by the parameter len. If the length of the message is too large to be sent all at once by the low-level protocol, -1 is returned, and errno is set to EMSGSIZE.

If the length of the send message is greater than the length of the socket interface send buffer, the send call will normally block unless the socket interface is set to a non-blocking mode. In non-blocking mode -1 is returned and errno is set to EAGAIN. The Select call can determine whether more data can be sent.

The structure MSGHDR is defined in the header file <sys/socket.h>.

4) *Returned value*

The length of the sent data is returned on success, -1 is returned on error, and errno represents the value of the error type.

I. Ioctl()

1) *Head file*

#include <sys/ioctl.h>

2) *Prototype*

intioctl(int d, intrequest, ...);

3) *Description*

Ioctl( ) calls operate on the parameters of the underlying device. Parameter d is the file handle, and the parameter request determines the type and size of the back parameters. See the <sys/ioctl.h> for the macro definition used to describe the parameter request.

4) *Returned value*

0 is returned on success, -1 is returned on error, and errno represents the error type value.

J. Getsockopt()，setsockopt()

1) Head file

#include <sys/types.h>

#include<sys/socket.h>

2) Prototype

Intgetsockopt(int s, intlevel, intoptname, void *optval, socklen_t *optlen);

Intsetsockopt(int s, int level, intoptname, const void *optval, socklen_toptlen);

3) Description

The Getsockopt() and setsockopt() calls can operate on the options of the socket interface. Options exist at multiple protocol levels, but are always represented at the highest socket interface level. When socket interface options are setting, it must be specify the level name and option name. For the socket interface level option, the level is called SOL_SOCKET. For other levels of protocol, other protocol control Numbers are provided, such as the TCP protocol, and the level name must be the TCP series.

The parameters optval and optlen are used when setsockopt calls access option values. For the getsockopt calls, they are buffers that return the request option value; the optlen parameter starts with the size of the buffer optval, and returns with the buffer size of the actual return value. If no option value can be returned, the parameter optval is set to NULL.

The optname and option parameters are sent to the appropriate core protocol module for interpretation without explanation. In the header file <sys/socket.h>, there is a detailed definition of the socket interface level and option structure, and the option formats and names for different protocol levels vary greatly.

Most interface-level options take an integer value as the parameter optval, and for setsockopt calls, the parameter must be non-zero to support Boolean options, or zero to disable.

In the design of IPv9 stream label, the following call can be used:

```
int on = 1；
struct in9_flowlabel_req freq；
structin9_addrdst_addr；
memcpy(&(freq.flr_dst)，&dst_addr，32)；
freq.flr_label = htonl(0x0000000f)；
freq.flr_action = IPV9_FL_A_GET；
freq.flr_share = IPV9_FL_S_EXCL；
freq.flr_flags = IPV9_FL_F_CREATE；
freq.flr_expires = 0；
freq.flr_linger = 0；
freq.__flr_pad = 0；
setsockopt(s, IPPROTO_IPV9, IPV9_FLOWINFO_SEND, &on, sizeof(int))；
setsockopt(s, IPPROTO_IPV9, IPV9_FLOWINFO, &on, sizeof(int))；
setsockopt(s, IPPROTO_IPV9, IPV9_FLOWLABEL_MGR, &freq,
sizeof(structin9_flowlabel_req));
```

The above code sets the stream label of socket s to 0000f, where the destination address of the stream label is defined in dst_addr.

Structure in9 flowlabelreq is defined as follows

```
struct in9_flowlabel_req{
    struct in9_addrflr_dst；
    __u32    flr_label；
    __u8     flr_action；
    __u8     flr_share；
    __u16    flr_flags；
    __u16    flr_expires；
    __u16    flr_linger；
```

```
        __u32      __flr_pad;
    };
```

4) Returned value

0 is returned on success, -1 is returned on error, and errno represents the error type value.

III. IPv9 Development instruction

1) Development environment

Centos7 operating system with Linux operating environment with IPv9 kernel;

VMware virtual machine image:: Centos7_ IPv9_ dev_vm.

The compiled program copy in Centos7_IPv9_dev_vm virtual machine image, it can run normally, provides the virtual machine application development and compilation environment, C language headers file and IPv9_Linux kernel.

2) IPv9 network application development directory:

/develop9
Development document directory:
/develop9/docs
The demo directory:
/develop9/test9
demo README
/develop9/test9/README

3) Test9 program

The test9 program mainly changes the socket family program file.

cd /develop9/test9

make

4) Demo operation

#Configure the IPv9 address

ifconfig9 eth1 add 32768[86[21[4]10001

#Start the IPv9 server program: /test9_tcpserver

#Start the IPv9 client program: /test9_tcpcli 32768[86[21[4]10001

Verify the caught: tcpdump -s 0 -i any -w t.cap, or wireshark with ipv9 plugin open t.cap.

IV. Conclusion

This paper introduces the commonly used socket able and interface functions, including creating a socket, binding function, link function, monitoring function and accept function, read the function and writing function, etc., each function is connected the header files, prototyping, description, and the return value, these are the basis of network programming, mastering these functions, which plays a major role for application development.

References

[1] https://zh.wikipedia.org/wiki/Berkeley%E5%A5%97%E6%8E%A5%E5%AD%97, Wikipedia: Berkeley sockets 2011-02-18, (Goodheart 1994, p. 11), (Goodheart 1994, p. 17)

[2] Cisco Networking Academy Program, CCNA 1 and 2 Companion Guide Revised

[3] Third Edition, P.480, ISBN 1-58713-150-1

[4] Jack Wallen (2019-01-22). "An Introduction to the ss Command".

[5] V. S. Bagad, I. A. Dhotre (2008), Computer Networks (5th revised edition, 2010 ed.), Technical Publications Pune, p. 52

[6] Ian Griffiths for IanG on Tap. 12 August, 2004. Raw Sockets Gone in XP

[7] "raw(7): IPv4 raw sockets - Linux man page". die.net.

[8] "Raw IP Networking FAQ". faqs.org.

[9] www-306.ibm.com - AnyNet Guide to Sockets over SNA

[10] books.google.com - UNIX Network Programming: The sockets networking API

[11] books.google.com - Designing BSD Rootkits: An Introduction to Kernel Hacking

[12] historyofcomputercommunications.info - Book: 9.8 TCP/IP and XNS 1981 - 1983

[13] mit.edu - The Desktop Computer as a Network Participant.pdf 1985