Linux socket

不积跬步 无以至千里 不积小流 无以成江海

zhaoyanming (A)

2020

# 基础知识：

## Socket函数详解:

### 头文件：

|  |
| --- |
| <sys/socket.h> |

### 函数原型：

|  |
| --- |
| int socket(int af, int type, int protocol); |

### 参数详解：

1. af：地址族(Address Familly),也就是IP地址类型
   * 常用网络类型AF\_INTE(IPv4地址)和AF\_INTE6(IPv6地址)；也可以使用PF\_INTE(IPv4地址)和PF\_INTE6(IPv6地址)。
   * 常用的进程间通信类型(本地类型) AF\_UNIX和AF\_LOCAL也可以使用PF\_UNIX和PF\_LOCAL
2. type：数据传输类型/套接字类型，常用的类型SOCKET\_STREAM(流格式套接字/面向连接)和SOCKET\_DGRAM(数据报套接字/无连接套接字)。
3. protocol：传输协议，常用的类型IPPROTO\_TCP(TCP传输协议)和IPPROTO\_UDP(UDP传输协议)；设置为“0”时表示系统自动匹配传输协议。

### 套接字特征：

* SOCKET\_STREAM套接字特征：
  1. 数据在传输过程中不会丢失；
  2. 数据按照顺序进行传输；
  3. 数据发送和接受不是同步的(流格式套接字内部有一个字符数组的缓冲区)
* SOCKET\_DGRAM套接字特征：
  1. 快速传输但不保证传输顺序；
  2. 传输数据可能丢失和损坏(小概率事件)；
  3. 每次传输数据有大小限制；
  4. 数据发送和接受是同步的(发送一个数据，必须接受完后才可以发送下一条数据)

### 返回值：

文件描述符

### 创建套接字示例：

|  |
| --- |
| int ret = socket(AF\_INET, SOCKET\_DGRAM, IPPTORO\_UDP);//创建UDP传输的套接字  int ret = socket(AF\_INET, SOCKET\_STREAM, IPPTORO\_TCP);//创建TCP传输的套接字  int ret = socket(AF\_INET, SOCKET\_DGRAM, 0);// 创建UDP传输的套接字  int ret = socket(AF\_INET, SOCKET\_STREAM, 0); 创建TCP传输的套接字 |

## Bind函数详解：

### 函数原型：

|  |
| --- |
| int bind(int sock, struct sockaddr \*addr, socklen\_t addrlen); |

### 参数详解：

1. sock：socket() 返回的文件描述符；
2. addr：地址、端口号等；
3. addrlen：adrr参数的长度；
   * 数据结构sockaddr详解：

|  |
| --- |
| 1. **struct** sockaddr{ 2. sa\_family\_t sin\_family; //地址族（Address Family），也就是地址类型 3. char sa\_data[14]; //IP地址和端口号 4. }; |

一般编程中并不直接针对此数据结构操作，而是使用另一个与sockaddr等价的数据结构。

* + 数据结构sockaddr\_in详解(网络套接字)：

头文件: #include <sys/in.h>

|  |
| --- |
| 1. **struct** sockaddr\_in{ 2. sa\_family\_t sin\_family; //地址族（Address Family），也就是地址类型 3. uint16\_t sin\_port; //16位的端口号 4. **struct** in\_addr sin\_addr; //32位IP地址 5. char sin\_zero[8]; //不使用，一般用0填充 6. };   struct in\_addr{  in\_addr\_t s\_addr; //32位的IP地址  };  typedef struct in\_addr { union {             struct{                         unsigned char s\_b1,                         s\_b2,                         s\_b3,                         s\_b4;                         } S\_un\_b;            struct {                         unsigned short s\_w1,                         s\_w2;                         } S\_un\_w;             unsigned long S\_addr;           } S\_un; } IN\_ADDR; |

* + 数据结构sockaddr\_un详解(本地套接字)：

进程间通信的一种方式是使用UNIX套接字，人们在使用这种方式时往往用的不是网络套接字，而是一种称为本地套接字的方式。这样做可以避免为黑客留下后门。

头文件：#include <sys/un.h>

|  |
| --- |
| #define UNIX\_PATH\_MAX 108  struct sockaddr\_un {  \_\_kernel\_sa\_family\_t sun\_family; /\* AF\_UNIX \*/  char sun\_path[UNIX\_PATH\_MAX]; /\* pathname \*/  }; |

### 返回值：

0：成功 -1：失败

### 示例：

|  |
| --- |
| //创建套接字  int serv\_sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  //创建sockaddr\_in结构体变量  struct sockaddr\_in serv\_addr;  memset(&serv\_addr, 0, sizeof(serv\_addr)); //每个字节都用0填充  serv\_addr.sin\_family = AF\_INET; //使用IPv4地址  serv\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //具体的IP地址  serv\_addr.sin\_port = htons(1234); //端口  //将套接字和IP、端口绑定  bind(serv\_sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)); |

## Select函数：

### 头文件：

|  |
| --- |
| #include <sys/select.h> |

### 函数原型：

|  |
| --- |
| int select(int maxfdp1, fd\_set \*readset, fd\_set \*writeset, fd\_set \*exceptset,struct timeval \*timeout); |

### 函数作用：

在Linux中，我们可以使用select函数实现I/O端口的复用，传递给 select函数的参数会告诉内核：

* 我们所关心的文件描述符
* 对每个描述符，我们所关心的状态。(我们是要想从一个文件描述符中读或者写，还是关注一个描述符中是否出现异常)
* 我们要等待多长时间。(我们可以等待无限长的时间，等待固定的一段时间，或者根本就不等待)

从 select函数返回后，内核告诉我们一下信息：

* 对我们的要求已经做好准备的描述符的个数
* 对于三种条件哪些描述符已经做好准备.(读，写，异常)

有了这些返回信息，我们可以调用合适的I/O函数(通常是 read 或 write)，并且这些函数不会再阻塞

### 参数详解：

1. maxfdp1：是一个整数值，是指集合中所有文件描述符的范围，即所有文件描述符的最大值加1，不能错；

说明：对于这个原理的解释可以看上边fd\_set的详细解释，fd\_set是以位图的形式来存储这些文件描述符。maxfdp也就是定义了位图中有效的位的个数。

1. \*readset：标明是否从文件中读取数据；
2. \*writeset：标明是否从文件中写入数据；
3. \*exceptset：监视文件错误异常；
4. \*timeout：select的超时时间；
   * NULL值：将select置于阻塞状态，一定等到监视文件描述符集合中某个文件描述符发生变化为止；
   * 将时间值设为0秒0毫秒，就变成一个纯粹的非阻塞函数，不管文件描述符是否有变化，都立刻返回继续执行，文件无变化返回0，有变化返回一个正值；
   * timeout的值大于0，这就是等待的超时时间，即 select在timeout时间内阻塞，超时时间之内有事件到来就返回了，否则在超时后不管怎样一定返回，返回值同上述；

### 宏函数：

|  |
| --- |
| int FD\_ZERO(int fd, fd\_set \*fdset); //将一个fd\_set类型变量的所有位都设为0  int FD\_CLR(int fd, fd\_set \*fdset); //将一个fd\_set类型变量的某位都设为0  int FD\_SET(int fd, fd\_set \*fd\_set); //修改fd\_set类型变量的某位  int FD\_ISSET(int fd, fd\_set \*fdset); //判断fd\_set类型变量的某位是否被重置 |

### 示例：

|  |
| --- |
| #include <sys/time.h>  #include <stdio.h>  #include <fcntl.h>  int main()  {  int fd\_key, ret;  char value;  fd\_set readfd;  struct timeval timeout;  fd\_key = open("/dev/tty", O\_RDONLY);  timeout.tv\_sec=0;  timeout.tv\_usec=0;  while(1){  FD\_ZERO(&readfd); /\* 清空文件描述符集合 \*/  FD\_SET(fd\_key, &readfd); /\* 添加文件描述符到集合 \*/  ret = select(fd\_key + 1, &readfd, NULL, NULL, &timeout);  if(FD\_ISSET(fd\_key, &readfd)){ /\* 测试fd\_key是否在描述符集合中 \*/  read(fd\_key, &value, 1);  if('\n' == value){  continue;  }  }  }  } |

## recvfrom函数:

### 头文件：

|  |
| --- |
| #include <sys/socket.h> |

### 函数原型：

|  |
| --- |
| ssize\_t recvfrom(int sock, void \*buf, size\_t len, int flags,struct sockaddr \*from, socklen\_t \*fromlen); |

### 函数作用：

从套接字上接收一个消息，可同时应用于面向连接的和无连接的套接字。

### 参数详解：

1. sock：索引将要从其接收数据的套接字；
2. buf：存放消息接收后的缓冲区；
3. len：buf所指缓冲区的容量；
4. flags：是以下一个或者多个标志的组合体，可通过or操作连在一起；

|  |
| --- |
| #define SO\_EE\_ORIGIN\_NONE 0  #define SO\_EE\_ORIGIN\_LOCAL 1  #define SO\_EE\_ORIGIN\_ICMP 2  #define SO\_EE\_ORIGIN\_ICMP6 3 |

1. from：指向存放对端地址的区域，如果为NULL，不储存对端地址；
2. 作为入口参数，指向存放表示from最大容量的内存单元。作为出口参数，指向存放表示from实际长度的内存单元；

### 返回值：

成功执行时，返回接收到的字节数。另一端已关闭则返回0。失败返回-1

附：Errorno详解：

|  |
| --- |
| EAGAIN：套接字已标记为非阻塞，而接收操作被阻塞或者接收超时  EBADF：sock不是有效的描述词  ECONNREFUSE：远程主机阻绝网络连接  EFAULT：内存空间访问出错  EINTR：操作被信号中断  EINVAL：参数无效  ENOMEM：内存不足  ENOTCONN：与面向连接关联的套接字尚未被连接上  ENOTSOCK：sock索引的不是套接字 |

### 与recv对比：

* recv一般只用在面向连接的套接字；
* recvfrom可同时应用于面向连接的和无连接的套接字

## sendto函数：

### 头文件：

|  |
| --- |
| #include <sys/socket.h> |

### 函数原型：

|  |
| --- |
| ssize\_t sendto(int sock, const void \*buf, size\_t len, int flags, const struct sockaddr \*to, socklen\_t tolen); |

### 函数作用：

发送消息(可用于无连接的套接字，也可用于基于连接的套接字)

### 参数详解：

1. sock：索引将要从其发送数据的套接字；
2. buf：指向将要发送数据的缓冲区；
3. len：以上缓冲区的长度；
4. to：指向存放接收端地址的区域，可以为NULL；
5. tolen：以上内存区的长度，可以为0；

### 返回值：

成功执行时，返回已发送的字节数。失败返回-1

附：Errorno详解：

|  |
| --- |
| EACCES：对于Unix域套接字，不允许对目标套接字文件进行写，或者路径前驱的一个目录节点不可搜索  EAGAIN，EWOULDBLOCK： 套接字已标记为非阻塞，而发送操作被阻塞  EBADF：sock不是有效的描述词  ECONNRESET：连接被用户重置  EDESTADDRREQ：套接字不处于连接模式，没有指定对端地址  EFAULT：内存空间访问出错  EINTR：操作被信号中断  EINVAL：参数无效  EISCONN：基于连接的套接字已被连接上，同时指定接收对象  EMSGSIZE：消息太大  ENOMEM：内存不足  ENOTCONN：套接字尚未连接，目标没有给出  ENOTSOCK：sock索引的不是套接字  EPIPE：本地连接已关闭 |

### sendto与send解析：

同recv与recvfrom