套接字的默认状态是阻塞的，这就意味着当发出一个不能立即完成的套接字调用时，其进程将投入休眠，等待响应操作完成。可能阻塞的套接字分为以下四类：（1）输入操作read、readv、recv、recvform和recvmsg;（2）输出操作write、writev、send、sendto和sendmsg;（3）发起外出连接connect;（4）接受外来连接accept。

**1、非阻塞读**

如果某个进程对一个阻塞的TCP套接字调用读操作，并且改套接字的接受缓冲区中么哎呦数据可读，该进程会进入睡眠直到有一些数据到达。

TCP是字节流协议，一旦缓冲区有一些数据达到就能被唤醒，这些数据可以是单个字节，也可以是一个完整的TCP分节数据。如果想等到某个固定数目的数据可读为止，可以使用循环读取或者指定MSG\_WAITWALL函数。

UDP是数据报协议，一个阻塞的UDP套接字在接收缓冲区为空时调用读函数，该进程将进入睡眠状态，直到有UDP数据报达到。

对非阻塞的套接字，如果读操作不能被满足（TCP套接字至少一个字节数据可读，UDP套接字有一个完整的数据报可读），相应调用会立即返回一个EWOULDBLOCK或EAGAIN错误。

**1.1 设置非阻塞读**

这里以套接字描述符为例，有两个方法。第一个，**使用setsockopt对套接字设置O\_NONBLOCK标志，后续对套接字的所有读写操作函数均非阻塞**；第二个，**调用带有MSG\_DONTWAIT标志的的读函数（如readv、recv、recvfrom、recvmsg等），临时将当前读调用设置为非阻塞**。

int setNonblocking(int fd, int enable)

{

int flags;

if (flags = fcntl(fd, F\_GETFL, 0) < 0){

perror("");

return -1;

}

// 开启或关闭

if (enable == 0) flags &= ~O\_NONBLOCK;

else flags |= O\_NONBLOCK;

if (fcntl(fd, F\_SETFL, flags) < 0){

perror("");

return -1;

}

return 0;

}

len = ::recvfrom(socket\_fd, buf, sizeof(buf), MSG\_DONTWAIT);

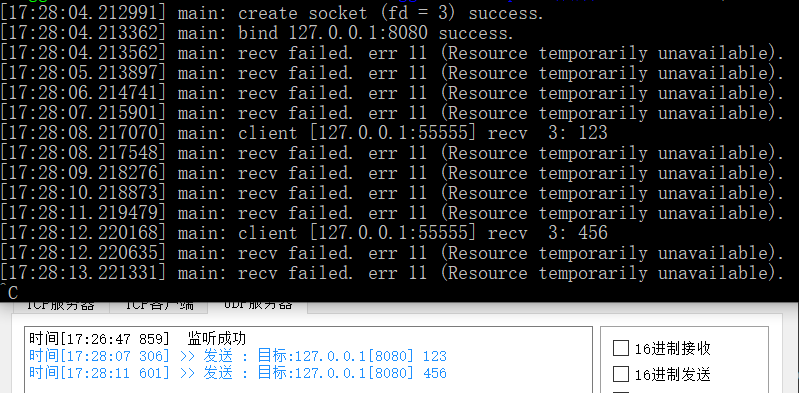
len = ::recvfrom(socket\_fd, buf, sizeof(buf), 0, (struct sockaddr \*)&clientaddr, &socklen);

**1.2 示例**

以UDP服务端为例，创建socket之后、调用setNonblocking函数，并bind，使用recvfrom进行读操作。由于设置非阻塞，每一次调用recvfrom将会立即返回，当返回值小于零且error为EAGAIN（wsl中为当前错误）不认为是错误，需要继续等待，并且为演示需要设定了1秒休眠时间。

|  |
| --- |
| while (1)  {  //int len = ::read(socket\_fd, buf, sizeof(buf));  //int len = ::recv(socket\_fd, buf, sizeof(buf), 0);  //int len = ::recvfrom(socket\_fd, buf, sizeof(buf), 0, NULL, NULL);  int len = ::recvfrom(socket\_fd, buf, sizeof(buf), 0, (struct sockaddr \*)&clientaddr, &socklen);  if (len < 0){  LOG("recv failed. err %d (%s).", errno, strerror(errno));  if (errno == EAGAIN){  sleep(1); // 演示需要，实际根据情况给一定的延时  continue;  }else{  break;  }  }else{  char ip[INET6\_ADDRSTRLEN];  inet\_ntop(clientaddr.sin\_family, &clientaddr.sin\_addr, ip, socklen);  int port = ntohs(clientaddr.sin\_port);  buf[len] = '\0';  LOG("client [%s:%d] recv %2d: %s", ip, port, len, buf);  }  //buf[len] = '\0';  if (strcmp(buf, "exit") == 0)  break;  } |

运行截图如下：



**2、非阻塞写**

对一个TCP套接字进行写调用，内核将从应用进程中的缓冲区复制数据到套接字的发送缓冲区。对于阻塞的套接字，如果其发送缓冲区没有空间，进程将进入睡眠直到有空间为止。

对一个非阻塞的TCP套接字，如果缓冲区没有空间，输出函数会立即返回一个EWOULDBLOCK或EAGGIN错误。如果发送缓冲区有一些空间，返回值将是内核能够复制到该缓冲区的字节数。

UDP套接字不存在真正的发送缓冲区。内核只是复制应用进程数据并沿协议栈向下传播，渐次冠以UDP首部和IP首部。因此对于一个阻塞的UDP套接字，输出函数调用将不会因与TCP套接字一样的原因而阻塞，不过有可能会因为其他的原因而阻塞。

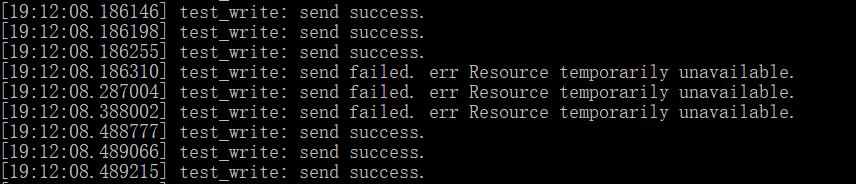
这里以非阻塞TCP客户端为例，使用带MSG\_DONTWAIT标志的send循环发送固定字节数据。由于默认的发送缓冲区较大，这里为模拟发送缓冲区满的异常情况，先使用setsockopt修改套接字发送区大小为100。

|  |
| --- |
| int opt = 100;  ret = setsockopt(socket\_fd, SOL\_SOCKET, SO\_SNDBUF, &opt, sizeof(opt));  if (ret < 0){  LOG("setsockopt SO\_SNDBUF failed.");  } |

循环发送数据代码部分为：

|  |
| --- |
| while (1)  {  len = ::send(socket\_fd, buf, 100, MSG\_DONTWAIT);  if (len < 0){  LOG("send failed. err %s.", strerror(errno));  if(errno == EAGAIN){  usleep(100\*1000); // 演示需要，实际根据情况给一定的延时  continue;  }else{  LOG("send failed. err %s.", strerror(errno));  break;  }  }else if(len == 0){  LOG("server closed.");  break;  }  LOG("send success.");  } |

运行部分截图如下



**3、非阻塞connect**

在上一章节中使用SIGALARM为connect设置超时，缩短连接时间。本节使用select指定以一个时间限制以缩短连接时间。

TCP连接的建立涉及一个三次握手过程，而且connect函数一直等到客户收到自己的SYN的ACK为止才返回。这意味着TCP的每个connect总是阻塞其调用进程至少一个到服务器的RTT时间。

如果对一个非阻塞的TCP套接字调用connect，并且连接不能立即建立，那么连接的建立能照样发起（譬如送出TCP三次握手的第一个分组），不过会返回一个EINPROGRESS错误。

|  |
| --- |
| static int connect\_nonb(int sockfd, const sockaddr\* pServAddr, socklen\_t socklen, int nsec)  {  int n, error = 0;  setNonblocking(sockfd); //设置套接字为非阻塞  if( (n=connect(sockfd, pServAddr, socklen)) < 0 ){  if(errno != EINPROGRESS) return -1;  }  if(n == 0)  goto done;  fd\_set rset, wset;  timeval tval;  FD\_ZERO(&rset);  FD\_SET(sockfd, &rset);  wset = rset;  tval.tv\_sec = nsec;  tval.tv\_usec = 0;  if( (n=select(sockfd+1, &rset, &wset, NULL, nsec ? &tval: NULL)) == 0){  close(sockfd);  error = ETIMEDOUT;  return -1;  }  if(FD\_ISSET(sockfd, &rset) || FD\_ISSET(sockfd, &wset)){  socklen\_t len = sizeof(error);  if( getsockopt(sockfd, SOL\_SOCKET, SO\_ERROR, &error, &len) < 0){  LOG("getsockopt failed. %s.", strerror(errno));  return -1;  }  }else{  LOG("select error: sock not set. %s.", strerror(errno));  return -1;  }  done:  setNonblocking(sockfd,0); // 清除非阻塞标志  if(error){  close(sockfd);  errno = error;  LOG("error: %s.", strerror(errno));  return -1;  }  return 0;  } |

套接字的各种实现和非阻塞connect会带来移植性问题。

1. 调用select之前可能连接已经建立并有对端数据到达。

这种情况下套接字没有错误，套接字可读可写，和建立失败下的套接字读写条件一致。需要通过getsockopt检查套接字上是否存在错误看来处理这种情况。

1. 不能假设套接字的可写（而不可读）条件是select返回套接字操作成唯一办法，需要要判断连接是否成功。
   1. 调用getpeername代替getsockopt。如果getpeername以ENOTCONN错误返回，那么建立失败，直接以SO\_ERROR调用getsockopt获取套接字上的待处理错误。
   2. 以值为0的长度参数调用read。如果read失败，说明connect失败，read返回的error给出连接错误原因。如果连接建立成功，read会返回0。
   3. 再调用一次connet，如果失败且返回错误是EISCONN，说明套接字已经连接成功。

**4、非阻塞accept**

如果对一个阻塞的套接字调用accept函数，并且尚无新的连接到达，调用进程将被投入睡眠。如果对一个非阻塞的套接字调用accept函数，并且尚无新的连接到达，accept调用将立即返回一个EWOULDBLOCK错误。

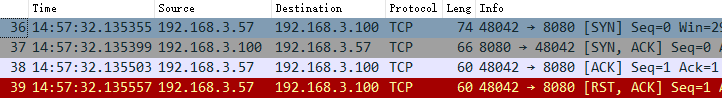
当使用select在某个监听套接字上等待一个外来连接，就没有必要把该套接字设置为非阻塞，因为如果select告诉该套接字上有连接就绪，那么随后的accept调用就不应该阻塞。但是当服务器繁忙时，无法在select之后立即调用accept，如果这端时间中客户终止某个连接（如客户端主动发送RST消息），就会出现问题。

**4.1 客户端建立连接立即断开**

客户端主函数代码：

|  |
| --- |
| static int client\_RST()  {  /// 1、创建socket  int socket\_fd = socket(AF\_INET,SOCK\_STREAM, 0); // tcp  if (socket\_fd == -1){  LOG("create socket failed. %s", strerror(errno));  return 1;  }else{  LOG("create socket (fd = %d) success.", socket\_fd);  }  /// 2、连接服务器  sockaddr\_in servaddr;  servaddr.sin\_family = AF\_INET;  inet\_pton(servaddr.sin\_family, SRV\_ADDR, &servaddr.sin\_addr);  servaddr.sin\_port = htons(SRV\_PORT);  int ret = ::connect(socket\_fd, (const sockaddr \*)&servaddr, sizeof(servaddr));  if (ret == -1){  LOG("connect %s:%d failed. %s", SRV\_ADDR, SRV\_PORT, strerror(errno));  return 1;  }else{  LOG("connect %s:%d success.", SRV\_ADDR, SRV\_PORT);  }  // 3、模拟的中止连接，发送RST  struct linger ling;  ling.l\_onoff = 1;  ling.l\_linger = 0;  // wsl 18.04实测设置成功后.调用close不发送RST（ubuntu16.04 实体机可行）  ret = setsockopt(socket\_fd, SOL\_SOCKET, SO\_LINGER, &ling, sizeof(ling));  if (ret < 0){  LOG("setsockopt SO\_LINGER failed. %s", strerror(errno));  }  close(socket\_fd); // 不能继续发送和读写; 发送RST到服务端; 不进入TIME\_WAIT状态  return 0;  } |

运行后，使用wireshark抓包结果如下



建立连接经过三次握手，使用SO\_LINGER参数调用setsockopt之后调用close关闭连接，将不会经过四次挥手而仅仅发送一个RST消息。

**4.2 服务端使用非阻塞accept**

模拟本节开头流程，步骤如下：

1. 客户建立一个连接并随后终止它
2. select向服务器返回可读条件，不过服务器过一小段时间再调用accept
3. 服务器从select返回到调用accept期间，服务器TCP收到收到客户的RST
4. 已完成的连接被服务器TCP驱除队列，假设队列中没有其他已完成连接
5. 服务器调用accept，由于没有任何已完成的连接，服务器于是阻塞

服务器会一直阻塞在accept调用上，直到某个其他客户建立一个连接为止。这种情况类似DDOS攻击，不过服务器单纯阻塞在accept上，无法处理其他任何已就绪的描述符。

解决办法步骤如下：

（1）使用select获悉某个监听套接字上有已完成连接且准备好被accept时，总是把这个监听套接字设置为非阻塞。

（2）在后续的accept中忽略以下错误：EWOULDBLOCK、ECONNABORTED、EPROTO（不同平台下客户端终止连接返回的错误）和EINTR（异常信号被捕获）。

服务端主函数代码

|  |
| --- |
| static int server\_accept()  {  /// 1、创建socket  int socket\_fd = socket(AF\_INET,SOCK\_STREAM, 0); // tcp  if (socket\_fd == -1){  LOG("create socket failed. %s", strerror(errno));  return 1;  }else{  LOG("create socket (fd = %d) success.", socket\_fd);  }  /// 2、绑定到本地端口  sockaddr\_in servaddr;  servaddr.sin\_family = AF\_INET;  inet\_pton(servaddr.sin\_family, SRV\_ADDR, &servaddr.sin\_addr);  servaddr.sin\_port = htons(SRV\_PORT);  int ret = ::bind(socket\_fd, (const sockaddr \*)&servaddr, sizeof(servaddr));  if (ret == -1){  LOG("bind %s:%d failed. %s.", SRV\_ADDR, SRV\_PORT, strerror(errno));  return 1;  }else{  LOG("bind %s:%d success.", SRV\_ADDR, SRV\_PORT);}  /// 3、监听  ret = ::listen(socket\_fd,5);  if(ret == -1){  LOG("listen failed. %s", strerror(errno));  return 1;  }else{  LOG("listening ...");  }  /// 4、等待连接 nonblocking accept  {  fd\_set rset;  timeval tval;  while(true)  {  FD\_ZERO(&rset);  FD\_SET(socket\_fd, &rset);  tval.tv\_sec = 5;  tval.tv\_usec = 0;  int n=select(socket\_fd+1, &rset, NULL, NULL, &tval);  if(n == 0){  errno = ETIMEDOUT;  usleep(1000);  continue;  }  else{  if(FD\_ISSET(socket\_fd, &rset)){ // 新的连接  LOG("listening socket readable.");  sleep(5);  setNonblocking(socket\_fd,1); //开启非阻塞, 否则当前accept可能阻塞  sockaddr\_in clientaddr;  socklen\_t socklen = sizeof(clientaddr);  int sock\_id = ::accept(socket\_fd, (sockaddr\*)&clientaddr, &socklen);  if(sock\_id < 0)  {  LOG("accept error. %s", strerror(errno));  if(errno == EAGAIN || errno == EWOULDBLOCK || errno == ECONNABORTED ||  errno == EPROTO || errno == EINTR)  {  LOG("accept ignore error. %s", strerror(errno));  continue;  }else{  return -1;  }  }else{  LOG("accept success.");  }    setNonblocking(socket\_fd,0); // 关闭非阻塞  char ip[INET6\_ADDRSTRLEN];  inet\_ntop(clientaddr.sin\_family, &clientaddr.sin\_addr, ip, socklen);  int port = ntohs(clientaddr.sin\_port);  LOG("accept client [%s:%d]", ip, port);  }else{  LOG("select other event.");  }  }  }  }  /// 5、关闭连接  ::close(socket\_fd);  return 0;  } |

在实际使用中单线程的服务端较少，基本上都会为每一个连接创建一个线程或者使用线程池进行管理，所以常规使用下不要考虑复杂，根据实际情况进行处理即可。