# **[阻塞与非阻塞的IO网络读写](https://www.cnblogs.com/charlesblc/p/6202402.html)**

看我之前的文章就知道，一般对于网络读的socket，都会加上O\_NONBLOCK，非阻塞的选项。

int setnonblocking(int fd) {

int old\_option = fcntl(fd, F\_GETFL);

int new\_option = old\_option | O\_NONBLOCK;

fcntl(fd, F\_SETFL, new\_option);

return old\_option;

}

为什么要加上呢。是为了效率。下面详细说一下阻塞和非阻塞。

基本概念：

阻塞IO: 必须做完IO操作才会返回。

非阻塞IO：操作成功与否，都会返回，需要通过其他方式判断具体操作是否成功。

## **读**

阻塞与非阻塞的区别：没有数据到达的时候，是否立刻返回。

读（read, recv, msgrcv）:

注意，这里的读，只是负责把数据从底层系统缓存copy到我们指定的位置。实际的数据到达是系统做的。

### 

### **阻塞情况（read, recv, msgrcv的行为）：**

1. 如果没有数据，会一直等待；

2. 有数据时候会读到用户指定的缓存区，但是如果数据量比较少，少于参数指定的大小，read也会立即返回，而不会一直等到数据足够。

阻塞读的原则：数据不超过指定长度的时候，有多少读多少，没有数据就会一直等待。

所以一般情况下，都需要采用循环读的方式，因为一次read不能保证读完需要的全部数据。

### 

### **非阻塞情况（read, recv, msgrcv的行为）**

1. 没有数据，就立即返回；

2. 有数据，也是采用有多少读多少的方式来处理。

所以，read完一次，要判断读到的数据长度或者错误码再决定是否再次读取。注意这里的EAGAIN错误码是需要继续读取，而返回0是对方已关闭连接。

[IMG_256](javascript:void(0);)

对非阻塞socket而言，EAGAIN不是一种错误。在VxWorks和Windows上，EAGAIN的名字叫做EWOULDBLOCK。错误信息为Resource temporarily unavailable，errno代码为11(EAGAIN)。

如果出现EINTR即errno为4，错误描述Interrupted system call，操作也应该继续。

EINTR指操作被中断唤醒，需要重新读/写。

而EAGAGIN不需要重新读/写已经操作的数据。

最后，如果recv的返回值为0，那表明连接已经断开，我们的接收操作也应该结束。

[IMG_257](javascript:void(0);)

综上，对于读而言，阻塞与非阻塞的区别在于，没有数据到达的时候是否立刻返回。

而recv函数有一个 MSG\_WAITALL的参数。

recv(sockfd, buff, buff\_size, MSG\_WAITALL);

这个参数意味着recv会争取等到数据填满buff\_size再返回，但是如果有中断的情况, recv还是会被大端，造成没有读完buff\_size的长度。

所以即使采用了recv+MSG\_WAITALL的方式，还是要循环读取，当然在大多数情况下是能读满的。

注意：MSG\_WAITALL只能在阻塞模式下使用，和非阻塞模式不能同时使用。

## **写**

写（write/send/msgsnd）的本质也是把用户态数据copy到系统底层去，然后由系统进行发送和实际写操作。只要完成了copy，就意味着写完成。

### **阻塞情况（write/send/msgsnd的行为）**

与阻塞读有多少读多少不同的是，阻塞写会一直阻塞，直到所有数据都完成，再返回。

这是因为，读的时候不知道需要读多少，防止一直等不到足够的数据；而写的时候是知道要写多少数据的。不过也可能被中断，大多数情况是能够写完的。

### **非阻塞情况（write/send/msgsnd的行为）**

非阻塞写，就是有多少写多少。能够写多少是根据本地网络拥塞情况为标准的，当网络拥塞严重的时候，网络层没有足够的内存来进行写操作，就会出现写不完的情况；这时候，阻塞写除非被中断，都会等到数据都写完；而非阻塞写，就是能写多少算多少。

### **IO模式设置方式**

### **Socket：**

方法1: 文章开始的方式，对flags加O\_NONBLOCK; （注：如果想设置成非阻塞，这样：flags&~O\_NONBLOCK）

方法2: recv, send函数的参数，最后一个参数设置成 MSG\_DONTWAIT，如下：

recv(sockfd, buff, buff\_size, MSG\_DONTWAIT);

send(sockfd, buff, buff\_size, MSG\_DONTWAIT);

对当次的函数，为非阻塞。

### **普通文件：**

方法1: open函数的第二个参数加上 O\_NONBLOCK，函数说明如下：

[IMG_258](javascript:void(0);)

#include <fcntl.h>int open(const char \*pathname, int oflag, ... /\* mode\_t mode \*/);

open函数用来打开或创建一个文件，若成功返回文件描述符，否则返回-1。

pathname是要打开或创建文件的名字。

oflag参数是下列一个或多个常量执行按位或运算的结果杀

O\_RDONLY　　只读打开

O\_WRONLY　 只写打开

O\_RDWR 读写打开

上面三个常量必须指定一个并且只能指定一个，下面一些常量则是可选的：

O\_APPEND　　将写入追加到文件的尾端

O\_CREAT 若文件不存在，则创建它。使用该选项时，需要第三个参数mode，用来指定新文件的访问权限位

O\_EXCL 如果同时指定了O\_CREAT，而文件已经存在，则会出错

O\_TRUNC 如果此文件存在，而且为只写或读写模式成功打开，则将其长度截短为0

O\_NOCTTY 如果pathname指的是终端设备，则不将该设备分配作为此进程的控制终端O\_NONBLOCK 如果pathname指的是一个FIFO文件、块设备文件或字符设备文件，则此选项将文件的本次打开操作和后续的I/O操作设置为非阻塞模式

[IMG_259](javascript:void(0);)

方法2，同socket的方法1，用F\_SETFL和flags|O\_NONBLOCK.

### **消息队列：**

msgsnd和msgrcv的最后一个参数加上 IPC\_NOWAIT：

[IMG_260](javascript:void(0);)

int msgsnd (int msqid, const void \*ptr, size\_t length, IPC\_NOWAIT) ;

参数flag的值可以指定为IPC\_NOWAIT。这类似于文件IO的非阻塞IO标志。若消息队列已满，则指定IPC\_NOWAIT使得msgsnd立即出错返回EAGAIN。

如果没有指定IPC\_NOWAIT，则进程阻塞直到下述情况出现为止：①有空间可以容纳要发送的消息 ②从系统中删除了此队列（返回EIDRM“标识符被删除”）③捕捉到一个信号,并从信号处理程序返回（返回EINTR）

ssize\_t msgrcv (int msqid, void\* ptr, size\_t length, long type, IPC\_NOWAIT) ;

参数ptr指定所接收消息的存放位置。参数length指定了数据部分大小(只想要多长的数据)

参数type指定希望从队列中读出什么样的消息。

type == 0 返回队列中的第一个消息

type > 0 返回队列中消息类型为type的第一个消息

type < 0 返回队列中消息类型值小于或等于type绝对值的消息，如果这种消息有若干个。则取类型值最小的消息。

（如果一个消息队列由多个客户进程和一个服务器进程使用，那么type字段可以用来包含客户进程的进程ID）

参数flag可以指定为IPC\_NOWAIT，使操作不阻塞。

[IMG_261](javascript:void(0);)

这类似于文件IO的非阻塞IO标志。比如msgsnd，若消息队列已满，则指定IPC\_NOWAIT使得msgsnd立即出错返回EAGAIN。