链滴

IO中同步、异步与阻塞、非阻塞的区别

作者：casablinca

原文链接：https://ld246.com/article/1453647673862

来源网站：链滴

<p>一、同步与异步
同步/异步, 它们是消息通知机制

1. 概念解释
A 同步
所谓同步，就是在发出一个功能调用时，在没有得到结果之前该调用就不返回。

按照这个定义，其实绝大多数函数都是步调用（例如sin isdigit等）。
但是一般而言，我们在说同步、异步的时候特指那些需要其他部件协作或者需要一定时间完成的任务。
最常见的例子就 SendMessage。
该函数发送一个消息给某个窗口，在对方处理完消息之前这个函数不返回。
当对方处理完毕以后，该函数才把消息处理函数所返回的返回给调用者。

B. 异步
异步的概念同步相对。
当一个异步过程调用发出后，调用者不会立刻得到结果。</span
实际处理这个调用的部件是在调用发出后，
通过状、通知来通知调用者，或通过回调函数处理这个调用。
<b />以 Socket为例，
当一个客户端通过调用 Connect函数发出一个接请求后，调用者线程不用等待结果，可立刻继续向下运行。
当连接真正建起来以后，socket底层会发送一个消息通知该对象。

C. 三种返回结途径
执行部件和调用者可以通过三种途径返回结果：
a.   状态、
b.   通知、
c.   回调函数。

可以使用哪一种依赖于执行部件的实现，除非执行部件提供多种选择否则不受调用者控制。

a. 如果执行部件用状态来通知，
    那么调用者就需要每隔一定时间检查一次，效率就很低
    有些初学多线程编程的人，总喜欢用一个循环去检查某个变量的值，这其实是种很严重的错误。

b. 如果是使用通知的方式，
    效率则很高，因为执行部件几乎不需要做额外的操作。

c. 至于回调函数，
    和通知没太多区别。

2. 举例说明
理解这两个念，可以用去银行办理业务(可以取钱，也可以存钱)来比喻:
当到银行后,
.可以去ATM机前排队等候   -- (排队等候)就是同步等待消息
.可以去大厅拿号,等到排我的号时，
 柜台的人会通知我轮到我去办理业务.   &bsp;   -- (等待别人通知)就是异步等待消息.

在异步消息通知机制中,
等待消息者(在这个例中就是等待办理业务的人)往往注册一个回调机制,
在所等待的事件被触发时触发机制(在这里是柜台的人)通过某种机制(在这里是写在小纸条上的号码)
到等待该事件的人.

在select/poll 等IO 多路复用机制中就是fd,
当消息被触发时,触发机制通过fd 找到处理该fd的处理函数.

<pan>3. 在实际的程序中,
同步消息通知机制：就好比单的read/write 操作,它们需要等待这两个操作成功才能返回;
                    同步, 是由处理消息者自己去等待消息是被触发;
异步消息通知机制：类似于select/poll 之类的多路复用IO 操作,
                  当所注的消息被触发时,由消息触发机制通知触发对消息的处理.
    nbsp;   异步, 由触发机制来通知处理消息者;

还是回到上面的例子,
轮到你办理业务, 这个是你关注的消息,
而办理什么业务, 就是对这个消息的处理,
<pan>两者是有区别的.

而在真实的IO 操作时: 所关注的消息就是 &nbp;   该fd是否可读写,
          &nbs ;   而对消息的处理是   对这个fd 进行读写.</pan>

同步/异步仅仅关注的是如何通知消息,它们对如何处理消息并不关心,</san>
好比说,银行的人仅仅通知你轮到你办理业务了,
而办理务什么业务(存钱还是取钱)他们是不知道的.

二、阻塞与非塞
阻塞/非阻塞, 它们是程序在等待消息(无所谓同步或者异步)时状态.

1. 概念解释
A.

塞
阻塞调用是指调用结果返回之前，当前线程会被挂起。函数只有在得到结之后才会返回。
有人也许会把阻塞调用和同步调用等同起来，实际上他是不的。
对于同步调用来说，很多时候当前线程还是激活的，只是从逻辑上当前数没有返回而已。

socket接收数据函数recv是一个阻塞调用的例子。/span>
当socket工作在阻塞模式的时候， 如果没有数据的情况下调用该函数，则当线程就会被挂起，直到有数据为止。

B. 非阻塞
<spa>非阻塞和阻塞的概念相对应，指在不能立刻得到结果之前，该函数不会阻塞当前线程，而立刻返回。

C. 对象的阻塞模式和阻塞函数调用
对象是否处于阻塞模式和函数是不是阻塞调用有很强的相关性，但是并不是一一对应的。

阻塞对象上可以有非阻塞的调用方式，我们可以通过一定的API轮询状态，
在适当的时候调用阻塞函数，就可以避免阻塞。<br /span>而对于非阻塞对象，调用特殊的函数也可以进入阻塞调用。函数select就是这样的一个例子。

2. 举例说明
继续上面的个例子,
不论是排队等待，还是使用号码等待通知,
果在这个等待的过程中,
. 等待者除了等待消息之外不能做其它的事情,那么该制就是阻塞的,
  表现在程序中,也就是该程序一直阻塞在该函数调用不能继续往下执行.
. 相反,有的人喜欢在银行办理这些业务的时候一边打打电发发短信一边等待,这样的状态就是非阻塞的,
  因为他(等待者)没有阻在这个消息通知上,而是一边做自己的事情一边等待.

三、混淆的点
很多人也会把异步和非阻塞混淆,
<spa>因为异步操作一般都不会在真正的IO 操作处被阻塞,
比如如果用select 函数当select 返回可读时再去read 一般都不会被阻塞
就好比当你的号码排到时般都是在你之前已经没有人了,所以你再去柜台办理业务就不会被阻塞.
可见,步/异步与阻塞/非阻塞是两组不同的概念,它们可以共存组合,

而很多之所以把同步和阻塞混淆,我想也是因为没有区分这两个概念,
比如阻塞的read write 操作中,其实是把消息通知和处理消息结合在了一起,
在这里所关注的消就是fd 是否可读/写,而处理消息则是对fd 读/写.
当我们将这个fd 设置为非阻的时候,read/write 操作就不会在等待消息通知这里阻塞,
如果fd 不可读/写操作立即返回.

四、同步/异步与阻塞/非阻塞的组合析
_____阻塞_____非阻塞____
<pan>同步 | 同步阻塞   同步非阻塞</span
异步 | 异步阻塞   异步非阻

同步阻塞形式:
  效率是最低的，</spn>
  拿上面的例子来说，就是你专心排队，什么别的事都不做。

  实际程序中
  就是未对fd 设置O_NONBLOK 标志位的read/write 操作,

异步阻塞形式:
nbsp; 如果在银行等待办理业务的人采用的是异步的方式去等待消息被触发,也就是领了一张小纸条,</pan>
  假如在这段时间里他不能离开银行做其它的事情,那么很显然,这个人被塞在了这个等待的操作上面;

  异步操作是可以被阻塞的,只不过它不是在处理消息时阻塞,而是在等待消息被触发时被阻塞.
  比如select 函数,
  假如传入的最后一个timeout 参数为NULL,那么如所关注的事件没有一个被触发,
  程序就会一直阻塞在这个select 调用.

同步非阻塞形式:
  实际上是效率低的,
  想象一下你一边打着电话一边还需要抬头看到底队伍排到你了没,
  如果把打电话和观察排队的位置看成是程序的两个操作的话,</spa>
  这个程序需要在这两种不同的行为之间来回的切换,效率可想而知是低下的;

  很多人会写阻塞的read/write 操作,
&bsp; 但是别忘了可以对fd 设置O_NONBLOCK 标志位,这样就可以将同步操作变成非阻塞的了;</spa>

异步非阻塞形式:
  效率更高,
  因为打电话是你(等待者)的事情,而通知你则是柜台(消息触发机制)的事情,<b /> 程序没有在两种不同的操作中来回切换.

  如说，这个人突然发觉自己烟瘾犯了,需要出去抽根烟,
  于是他告诉堂经理说,排到我这个号码的时候麻烦到外面通知我一下(注册一个回调函数),
nbsp; 那么他就没有被阻塞在这个等待的操作上面,自然这个就是异步+非阻塞的方式了.

  如果使用异步非阻塞的情况,
  比如aio_*组操作,当发起一个aio_read 操作时,函数会马上返回不会被阻塞,
  当所注的事件被触发时会调用之前注册的回调函数进行处理,</p>

原文链接：IO中同步、异步与阻塞、非阻塞的区别