

guisu，程序人生。逆水行舟，不进则退。

能干的人解决问题。智慧的人绕开问题(A clever person solves a problem. A wise person avoids it)

目录视图

摘要视图

RSS 订阅

友情链接

诚聘：
http://job.csdn.net/job/index?jobID=81336

个人资料



真实的归宿

访问：2156523次

积分：19009

等级：

BLOG

7

排名：第180名

原创：206篇 转载：0篇

译文：0篇 评论：808条

- 文章分类
- 操作系统 (5)

Linux (20)

MySQL (12)

PHP (42)

PHP内核 (11)

技术人生 (7)

数据结构与算法 (29)

云计算hadoop (24)

网络知识 (7)

c/c++ (23)

memcache (5)

HipHop (2)

计算机原理 (4)

Java (7)

socket网络编程 (8)

设计模式 (26)

AOP (2)

重构 (11)

重构与模式 (1)

大数据处理 (12)

搜索引擎Search Engine (15)

HTML5 (1)

Android (1)

webserver (3)

NOSQL (7)

NOSQL Mongo (0)

分布式 (1)

数据结构与算法 xi (0)

协议 (1)

信息论的熵 (0)

关于php的libevent扩展的应用 (0)

libevent简单介绍 (0)

Markdown那么好，还不来试试

中国云计算大会最新议题

5月问答又送C币咯！

Hadoop实战高手速成宝典

socket阻塞与非阻塞，同步与异步、I/O模型

分类：c/c++ socket网络编程

2012-04-12 16:35

70633人阅读

评论(45)

收藏

举报

socket

sockets

windows

api

服务器

目录(?)

1. 概念理解

2. Linux下的五种IO模型

1. 阻塞IO模型

2. 非阻塞IO模型

3. IO复用模型

4. 信号驱动IO

5. 异步IO模型

6. 5个IO模型的比较

7. selectpolllepoll简介

socket阻塞与非阻塞，同步与异步

作者：huangguisu

1. 概念理解

在进行网络编程时，我们常常见到同步(Sync)/异步(Async)，阻塞(Block)/非阻塞(Unblock)四种调用方式：

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。也就是必须一件一件事做,等前一件做完了才能做下一件事。

例如普通B/S模式（同步）：提交请求->等待服务器处理->处理完毕返回 这个期间客户端浏览器不能干任何事

异步：

异步的概念和同步相对。当一个异步过程调用发出后，调用者不能立刻得到结果。实际处理这个调用的部件在完成时，通过状态、通知和回调来通知调用者。

例如 ajax请求（异步）：请求通过事件触发->服务器处理（这是浏览器仍然可以作其他事情）->处理完毕

阻塞

阻塞调用是指调用结果返回之前，当前线程会被挂起（线程进入非可执行状态，在这个状态下，cpu不会给线程分配时间片，即线程暂停运行）。函数只有在得到结果之后才会返回。

有人也许会把阻塞调用和同步调用等同起来，实际上他是不同的。对于同步调用来说，很多时候当前线程还是激活的，只是从逻辑上当前函数没有返回而已。例如，我们在socket中调用recv函数，如果缓冲池中没有数据，这个函数就会一直等待，直到有数据才返回。而此时，当前线程还会继续处理各种各样的消息。

快递的例子：比如到你某个时候到A楼一层（假如是内核缓冲区）取快递，但是你不知道快递什么时候过来，你也不能干别的事，只能死等着。但你可以睡觉（进程处于休眠状态），因为你知道快递把货送来时一定会给你打个电话（假定一定能叫醒你）。

非阻塞

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，而会立刻返回。

还是等快递的例子：如果用忙轮询的方法，每隔5分钟到A楼一层(内核缓冲区)去看快递来了没有。如果没来，立即返回。而快递来了，就放在A楼一层，等你去取。

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去轮询状态，在适当的时候调用阻塞函数，就可以避免阻塞。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。

1. 同步，就是我调用一个功能，该功能没有结束前，我死等结果。

文章存档

- 2015年04月 (1)
- 2015年01月 (2)
- 2014年10月 (1)
- 2014年09月 (1)
- 2014年08月 (2)

展开

阅读排行

- 八大排序算法 (126864)
- Mysql 多表联合查询效率 (109156)
- hbase安装配置（整合到 (70809)
- socket阻塞与非阻塞，同 (70574)
- Nginx工作原理和优化、i (57158)
- Hadoop集群配置（最全i (51759)
- 深入理解java异常处理机 (48855)
- 设计模式（十八）策略模 (48307)
- Java输入输出流 (46241)
- PageRank算法 (42557)

评论排行

- 八大排序算法 (49)
- socket阻塞与非阻塞，同 (45)
- 深入理解java异常处理机 (44)
- 设计模式（十八）策略模 (35)
- 硬盘的读写原理 (33)
- 海量数据处理算法—Bit-M (24)
- 设计模式（一）工厂模式 (24)
- UML图中类之间的关系:从 (23)
- PHP SOCKET编程 (22)
- HDFS写入和读取流程 (20)

推荐文章

- * 2015博文大赛
- *CSDN Markdown简明教程-基本使用
- *CSDN Markdown简明教程-快速上手
- *CSDN Markdown如何绘制UML图
- *CSDN Markdown使用LaTeX编写数学公式
- *CSDN Markdown扩展语法

最新评论

- 八大排序算法
真实的归宿: @lyg542133352: 恩，严谨。谢谢提醒
- UML图中类之间的关系:依赖,泛化
mayfla: 讲的很清楚，也找到了要找的问题。博主赞
- 深入理解java异常处理机制
pomelocc: 写得真好
- 深入理解java异常处理机制
pomelocc: 写得真好
- B-树和B+树的应用：数据搜索和
初学者--: SQL Server是不是和MySQL的一样？
- Redis应用场景
Frank_Gao: 写的很好！！！！
- 八大排序算法
ZYQblog1989: 您实现的插入算法稍微有点问题，其实只要从第

- 2. 异步，就是我调用一个功能，不需要知道该功能结果，该功能有结果后通知我（回调通知）
- 3. 阻塞，就是调用我（函数），我（函数）没有接收完数据或者没有得到结果之前，我不会返回。
- 4. 非阻塞，就是调用我（函数），我（函数）立即返回，通过select通知调用者

同步IO和异步IO的区别就在于：数据拷贝的时候进程是否阻塞！

阻塞IO和非阻塞IO的区别就在于：应用程序的调用是否立即返回！

对于举个简单c/s 模式：

同步：提交请求->等待服务器处理->处理完毕返回这个期间客户端浏览器不能干任何事
异步：请求通过事件触发->服务器处理（这是浏览器仍然可以作其他事情）->处理完毕
同步和异步都只针对本机SOCKET而言的。

同步和异步,阻塞和非阻塞,有些混用,其实它们完全不是一回事,而且它们修饰的对象也不相同。

阻塞和非阻塞是指当进程访问的数据如果尚未就绪,进程是否需要等待,简单说这相当于函数内部的实现区别,也就是未就绪时是直接返回还是等待就绪;

而同步和异步是指访问数据的机制,同步一般指主动请求并等待I/O操作完毕的方式,当数据就绪后在读写的时候必须阻塞(区别就绪与读写二个阶段,同步的读写必须阻塞),异步则指主动请求数据后便可以继续处理其它任务,随后等待I/O,操作完毕的通知,这可以使进程在数据读写时也不阻塞。(等待"通知")

node.js里面的描述：

线程在执行中如果遇到磁盘读写或网络通信（统称为I/O 操作），通常要耗费较长的时间，这时操作系统会剥夺这个线程的CPU 控制权，使其暂停执行，同时将资源让给其他的工作线程，这种线程调度方式称为 阻塞。当I/O 操作完毕时，操作系统将这个线程的阻塞状态解除，恢复其对CPU的控制权，令其继续执行。这种I/O 模式就是通常的同步式I/O（Synchronous I/O）或阻塞式I/O（Blocking I/O）。
相应地，异步式I/O（Asynchronous I/O）或非阻塞式I/O（Non-blocking I/O）则针对所有I/O 操作不采用阻塞的策略。当线程遇到I/O 操作时，不会以阻塞的方式等待I/O 操作的完成或数据的返回，而只是将I/O 请求发送给操作系统，继续执行下一条语句。当操作系统完成I/O 操作时，以事件的形式通知执行I/O 操作的线程，线程会在特定时候处理这个事件。为了处理异步I/O，线程必须有事件循环，不断地检查有没有未处理的事件，依次予以处理。阻塞模式下，一个线程只能处理一项任务，要想提高吞吐量必须通过多线程。而非阻塞模式下，一个线程永远在执行计算操作，"这个线程所使用的CPU 核心利用率永远是100%"，I/O 以事件的方式通知。"在阻塞模式下，多线程往往能提高系统吞吐量，因为一个线程阻塞时还有其他线程在工作，多线程可以让CPU 资源不被阻塞中的线程浪费。"而在非阻塞模式下，线程不会被I/O 阻塞，永远在利用CPU。多线程带来的好处仅仅是在多核CPU 的情况下利用更多的核，而Node.js 的单线程也能带来同样的好处。这就是为什么Node.js 使用了单线程、非阻塞的事件编程模式。

2. Linux下的五种I/O模型

- 1)阻塞I/O (blocking I/O)
- 2)非阻塞I/O (nonblocking I/O)
- 3) I/O复用(select 和poll) (I/O multiplexing)
- 4)信号驱动I/O (signal driven I/O (SIGIO))
- 5)异步I/O (asynchronous I/O (the POSIX aio_functions))

前四种都是同步，只有最后一种才是异步IO。

阻塞I/O模型：

简介：进程会一直阻塞，直到数据拷贝完成

应用程序调用一个IO函数，导致应用程序阻塞，等待数据准备好。如果数据没有准备好，一直等待....数据准备好了，从内核拷贝到用户空间,IO函数返回成功指示。

我们 第一次接触到的网络编程都是从 listen()、send()、recv()等接口开始的。使用这些接口可以很方便的构建服务器 / 客户机的模型。

阻塞I/O模型图：在调用recv()/recvfrom（）函数时，发生在内核中等待数据和复制数据的过程。

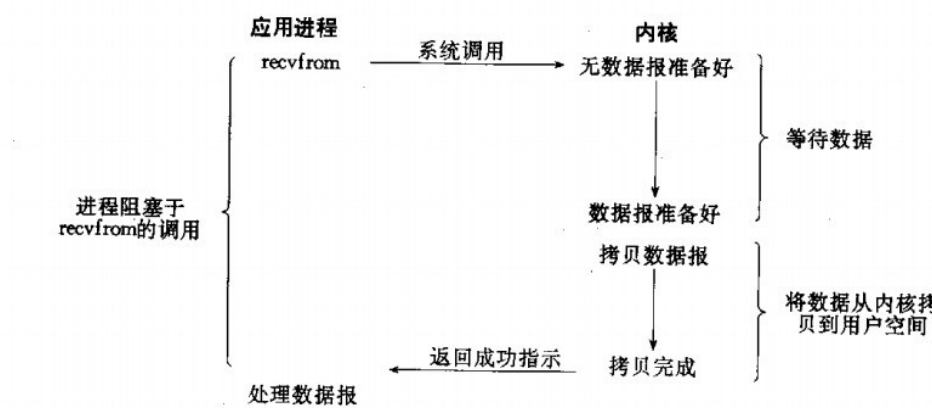


图 6.1 阻塞 I/O 模型

2个开始逐个与前面的比对，到头后再逐渐向后移动光标即可，如...

八大排序算法
ZYQblog1989: 确实，我跑了一遍发现这里缺少判断条件

设计模式（一）工厂模式Factory
light_and_darkness: 学习中，可惜是PHP的，想看C#的，PHP只能了解大概一点

Google 开源技术protobuf
DEMONU: @ms_X0828: 用protoc.exe

友情链接
图灵机器人：聊天api的最佳选择

当调用recv()函数时，系统首先查是否有准备好的数据。如果数据没有准备好，那么系统就处于等待状态。当数据准备好后，将数据从系统缓冲区复制到用户空间，然后该函数返回。在套接应用程序中，当调用recv()函数时，未必用户空间就已经存在数据，那么此时recv()函数就会处于等待状态。

当使用socket()函数和WSASocket()函数创建套接字时，默认的套接字都是阻塞的。这意味着当调用Windows Sockets API不能立即完成时，线程处于等待状态，直到操作完成。

并不是所有Windows Sockets API以阻塞套接字为参数调用都会发生阻塞。例如，以阻塞模式的套接字为参数调用bind()、listen()函数时，函数会立即返回。将可能阻塞套接字的Windows Sockets API调用分为以下四种：

- 1. 输入操作：recv()、recvfrom()、WSARecv()和WSARecvfrom()函数。以阻塞套接字为参数调用该函数接收数据。如果此时套接字缓冲区内没有数据可读，则调用线程在数据到来前一直睡眠。
- 2. 输出操作：send()、sendto()、WSASend()和WSASendto()函数。以阻塞套接字为参数调用该函数发送数据。如果套接字缓冲区没有可用空间，线程会一直睡眠，直到有空间。
- 3. 接受连接：accept()和WSAAccept()函数。以阻塞套接字为参数调用该函数，等待接受对方的连接请求。如果此时没有连接请求，线程就会进入睡眠状态。
- 4. 外出连接：connect()和WSAConnect()函数。对于TCP连接，客户端以阻塞套接字为参数，调用该函数向服务器发起连接。该函数在收到服务器的应答前，不会返回。这意味着TCP连接总会等待至少到服务器的一次往返时间。

使用阻塞模式的套接字，开发网络程序比较简单，容易实现。当希望能够立即发送和接收数据，且处理的套接字数量比较少的情況下，使用阻塞模式来开发网络程序比较合适。

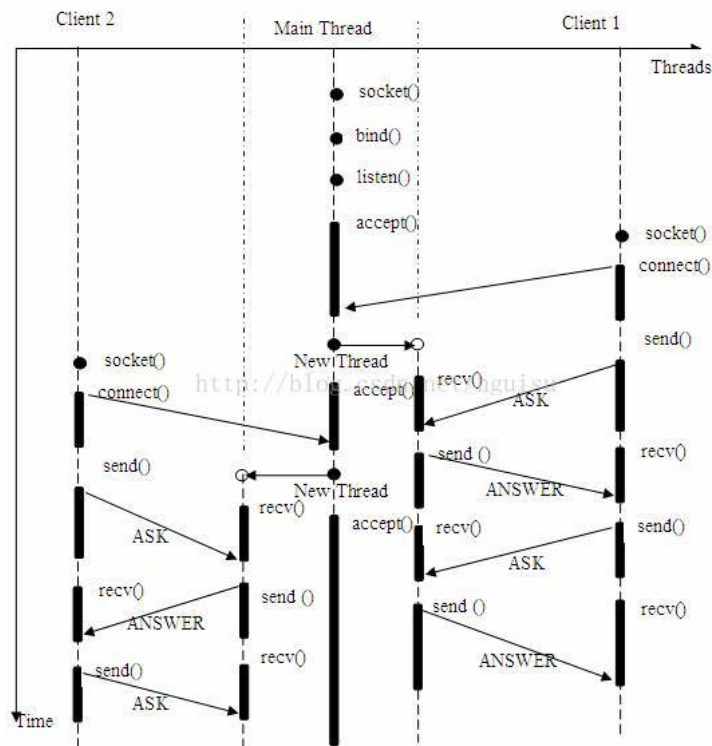
阻塞模式套接字的不足表现为，在大量建立好的套接字线程之间进行通信时比较困难。当使用“生产者-消费者”模型开发网络程序时，为每个套接字都分别分配一个读线程、一个处理数据线程和一个用于同步的事件，那么这样无疑加大系统的开销。其最大的缺点是当希望同时处理大量套接字时，将无从下手，其扩展性很差。

阻塞模式给网络编程带来了一个很大的问题，如在调用send()的同时，线程将被阻塞，在此期间，线程将无法执行任何运算或响应任何的网络请求。这给多客户机、多业务逻辑的网络编程带来了挑战。这时，我们可能会选择多线程的方式来解决这个问题。

应对多客户机的网络应用，最简单的解决方式是在服务器端使用多线程（或多进程）。多线程（或多进程）的目的是让每个连接都拥有独立的线程（或进程），这样任何一个连接的阻塞都不会影响其他的连接。

具体使用多进程还是多线程，并没有一个特定的模式。传统意义上，进程的开销要远远大于线程，所以，如果需要同时为较多的客户机提供服务，则不推荐使用多进程；如果单个服务执行体需要消耗较多的CPU资源，譬如需要进行大规模或长时间的数据运算或文件访问，则进程较为安全。通常，使用pthread_create()创建新线程，fork()创建新进程。

多线程/进程服务器同时为多个客户机提供应答服务。模型如下：



主线程持续等待客户端的连接请求，如果有连接，则创建新线程，并在新线程中提供为前例同样的问答服务。

上述多线程的服务器模型似乎完美的解决了为多个客户机提供问答服务的要求，但其实并不尽然。如果要同时响应成百上千路的连接请求，则无论多线程还是多进程都会严重占据系统资源，降低系统对外界响应效率，而线程与进程本身也更容易进入假死状态。

由此可能会考虑使用“线程池”或“连接池”。“线程池”旨在减少创建和销毁线程的频率，其维持一定合理数量的线程，并让空闲的线程重新承担新的执行任务。“连接池”维持连接的缓存池，尽量重用已有的连接、减少创建和关闭连接的频率。这两种技术都可以很好的降低系统开销，都被广泛应用很多大型系统，如apache，mysql数据库等。

但是，“线程池”和“连接池”技术也只是在一定程度上缓解了频繁调用 IO 接口带来的资源占用。而且，所谓“池”始终有其上限，当请求大大超过上限时，“池”构成的系统对外界的响应并不比没有池的时候效果好多少。所以使用“池”必须考虑其面临的响应规模，并根据响应规模调整“池”的大小。

对应上例中的所面临的可能同时出现的上千甚至上万次的客户端请求，“线程池”或“连接池”或许可以缓解部分压力，但是不能解决所有问题。

非阻塞IO模型：

简介：非阻塞IO通过进程反复调用IO函数（多次系统调用，并马上返回）；在数据拷贝的过程中，进程是阻塞的；

我们把一个SOCKET接口设置为非阻塞就是告诉内核，当所请求的I/O操作无法完成时，不要将进程睡眠，而是返回一个错误。操作系统函数将不断的测试数据是否已经准备好，如果没有准备好，继续测试，直到数据准备好为止。在这个不断测试的过程中，会大量的占用CPU的时间。

把SOCKET设置为非阻塞模式，即通知系统内核：在调用Windows Sockets API时，不要让线程睡眠，而应该让函数立即返回。在返回时，该函数返回一个错误代码。图所示，一个非阻塞模式套接字多次调用recv()函数的过程。前三次调用recv()函数时，内核数据还没有准备好。因此，该函数立即返回WSAEWOULDBLOCK错误代码。第四次调用recv()函数时，数据已经准备好，被复制到应用程序的缓冲区中，recv()函数返回成功指示，应用程序开始处理数据。

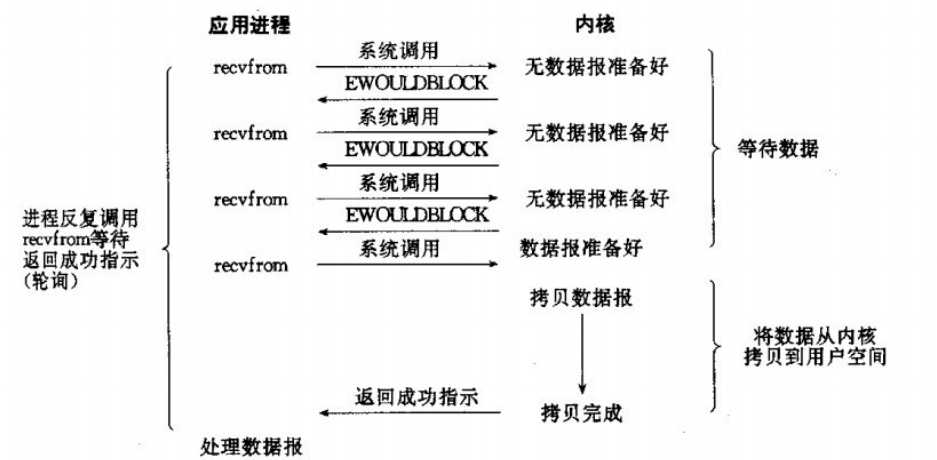


图 6.2 非阻塞 I/O 模型

当使用socket()函数和WSASocket()函数创建套接字时，默认都是阻塞的。在创建套接字之后，通过调用ioctlsocket()函数，将该套接字设置为非阻塞模式。Linux下的函数是:fcntl()。

套接字设置为非阻塞模式后，在调用Windows Sockets API函数时，调用函数会立即返回。大多数情况下，这些函数调用都会调用“失败”，并返回WSAEWOULDBLOCK错误代码。说明请求的操作在调用期间内没有时间完成。通常，应用程序需要重复调用该函数，直到获得成功返回代码。

需要说明的是并非所有的Windows Sockets API在非阻塞模式下调用，都会返回WSAEWOULDBLOCK错误。例如，以非阻塞模式的套接字为参数调用bind()函数时，就不会返回该错误代码。当然，在调用WSAStartup()函数时更不会返回该错误代码，因为该函数是应用程序第一调用的函数，当然不会返回这样的错误代码。

要将套接字设置为非阻塞模式，除了使用ioctlsocket()函数之外，还可以使用WSAAsyncselect()和WSAEventselect()函数。当调用该函数时，套接字会自动地设置为非阻塞方式。

由于使用非阻塞套接字在调用函数时，会经常返回WSAEWOULDBLOCK错误。所以在任何时候，都应仔细检查返回代码并作好对“失败”的准备。应用程序连续不断地调用这个函数，直到它返回成功指示为止。上面的程序清单中，在While循环体内不断地调用recv()函数，以读入1024个字节的数据。这种做法很浪费系统资源。

要完成这样的操作，有人使用MSG_PEEK标志调用recv()函数查看缓冲区中是否有数据可读。同样，这种方法也不好。因为该做法对系统造成的开销是很大的，并且应用程序至少要调用recv()函数两次，才能实际地读入数据。较好的做法是，使用套接字的“I/O模型”来判断非阻塞套接字是否可读可写。

非阻塞模式套接字与阻塞模式套接字相比，不容易使用。使用非阻塞模式套接字，需要编写更多的代码，以便在每个Windows Sockets API函数调用中，对收到的WSAEWOULDBLOCK错误进行处理。因此，非阻塞套接字便显得有些难于使用。

但是，非阻塞套接字在控制建立的多个连接，在数据的收发量不均，时间不定时，明显具有优势。这种套接字在使用上存在一定难度，但只要排除了这些困难，它在功能上还是非常强大的。通常情况下，可考虑使用套接字的“I/O模型”，它有助于应用程序通过异步方式，同时对一个或多个套接字的通信加以管理。

IO复用模型:

简介：主要是select和epoll；对一个IO端口，两次调用，两次返回，比阻塞IO并没有什么优越性；关键是能实现同时对多个IO端口进行监听；

I/O复用模型会用到select、poll、epoll函数，这几个函数也会使进程阻塞，但是和阻塞I/O所不同的的，这两个函数可以同时阻塞多个I/O操作。而且可以同时多个读操作，多个写操作的I/O函数进行检测，直到有数据可读或可写时，才真正调用I/O操作函数。

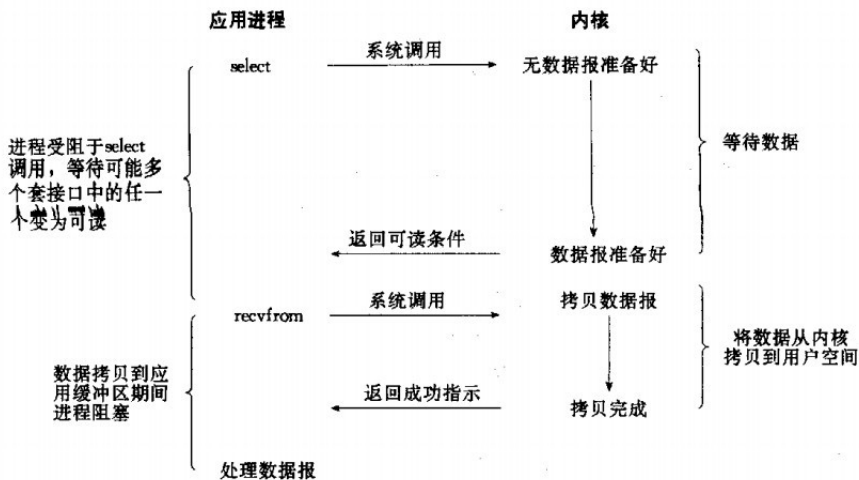


图 6.3 I/O 复用模型

信号驱动IO

简介：两次调用，两次返回；

首先我们允许套接口进行信号驱动I/O,并安装一个信号处理函数，进程继续运行并不阻塞。当数据准备好时，进程会收到一个SIGIO信号，可以在信号处理函数中调用I/O操作函数处理数据。

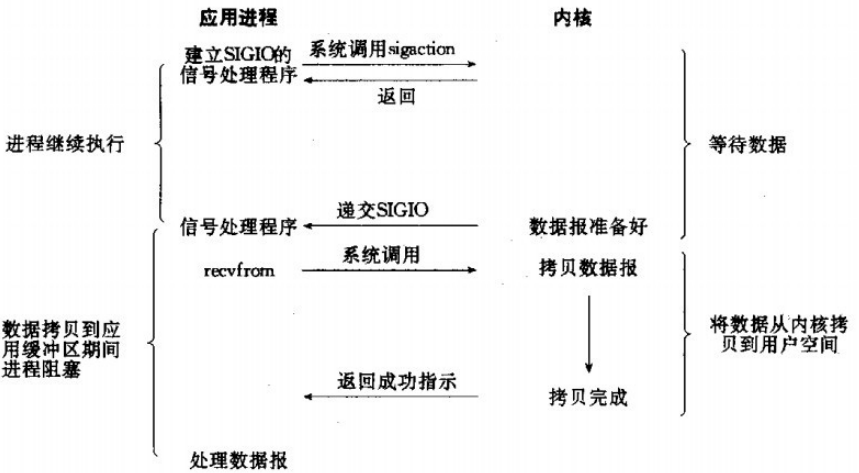
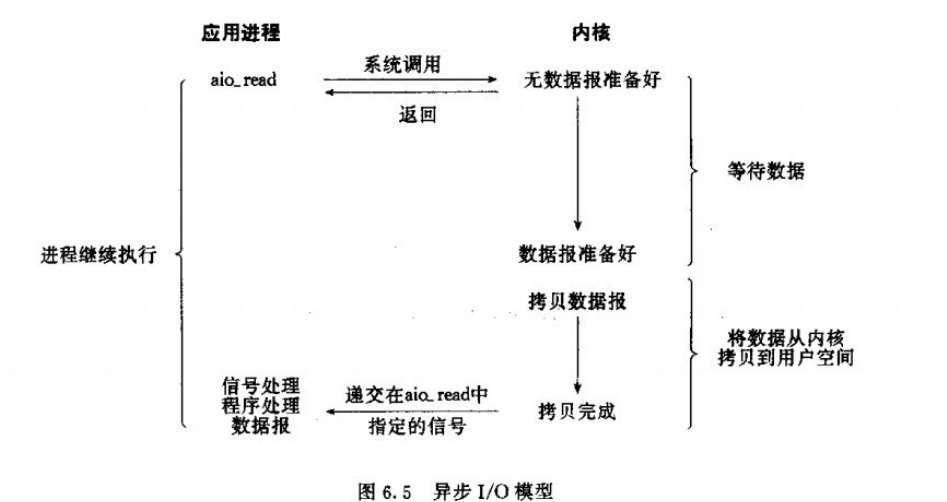


图 6.4 信号驱动 I/O 模型

异步IO模型

简介：数据拷贝的时候进程无需阻塞。

当一个异步过程调用发出后，调用者不能立刻得到结果。实际处理这个调用的部件在完成后，通过状态、通知和回调来通知调用者的输入输出操作

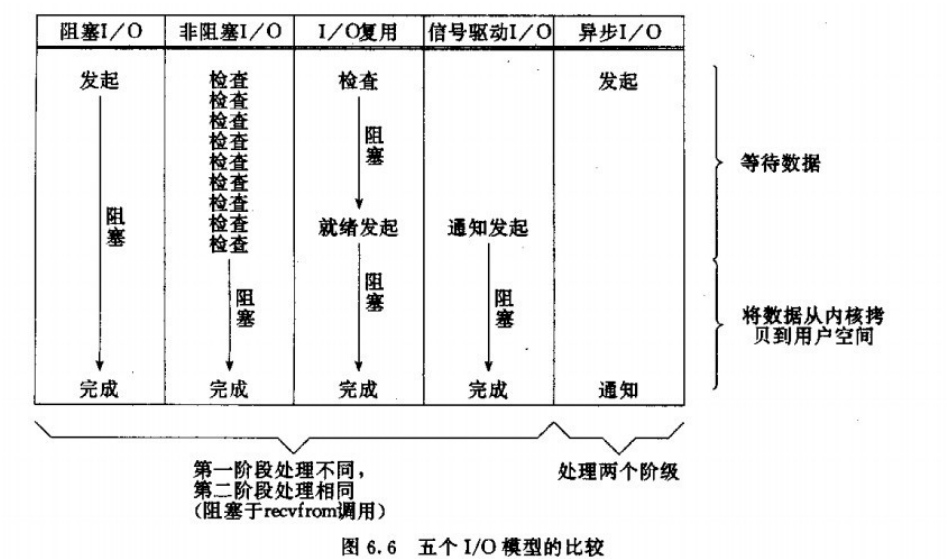


同步IO引起进程阻塞，直至IO操作完成。

异步IO不会引起进程阻塞。

IO复用是先通过select调用阻塞。

5个I/O模型的比较：



3. select、poll、epoll简介

epoll跟select都能提供多路I/O复用的解决方案。在现在的Linux内核里有都能够支持，其中epoll是Linux所特有，而select则应该是POSIX所规定，一般操作系统均有实现

select:

select本质上是通过设置或者检查存放fd标志位的数据结构来进行下一步处理。这样所带来的缺点是：

- 1、 单个进程可监视的fd数量被限制，即能监听端口的大小有限。

一般来说这个数目和系统内存关系很大，具体数目可以cat /proc/sys/fs/file-max察看。32位机默认是1024个。64位机默认是2048.
- 2、 对socket进行扫描时是线性扫描，即采用轮询的方法，效率较低：

当套接字比较多时，每次select()都要通过遍历FD_SETSIZE个Socket来完成调度,不管哪个Socket是活跃的,都遍历一遍。这会浪费很多CPU时间。如果能给套接字注册某个回调函数，当他们活跃时，自动完成相关操作，那就避免了轮询，这正是epoll与kqueue做的。
- 3、 需要维护一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大

poll:

poll本质上和select没有区别，它将用户传入的数组拷贝到内核空间，然后查询每个fd对应的设备状态，如果设备就绪则在设备等待队列中加入一项并继续遍历，如果遍历完所有fd后没有发现就绪设备，则挂起当前进程，直到设备就绪或者主动超时，被唤醒后它又要再次遍历fd。这个过程经历了多次无谓的遍历。

它没有最大连接数的限制，原因是它是基于链表来存储的，但是同样有一个缺点：

1、大量的fd的数组被整体复制于用户态和内核地址空间之间，而不管这样的复制是不是有意义。

2、poll还有一个特点是“水平触发”，如果报告了fd后，没有被处理，那么下次poll时会再次报告该fd。

epoll:

epoll支持水平触发和边缘触发，最大的特点在于边缘触发，它只告诉进程哪些fd刚刚变为就绪态，并且只会通知一次。还有一个特点是，epoll使用“事件”的就绪通知方式，通过epoll_ctl注册fd，一旦该fd就绪，内核就会采用类似callback的回调机制来激活该fd，epoll_wait便可以收到通知

epoll的优点：

1、没有最大并发连接的限制，能打开的FD的上限远大于1024（1G的内存上能监听约10万个端口）；

2、效率提升，不是轮询的方式，不会随着FD数目的增加效率下降。只有活跃可用的FD才会调用callback函数；
即Epoll最大的优点就在于它只管你“活跃”的连接，而跟连接总数无关，因此在实际的网络环境中，Epoll的效率就会远远高于select和poll。

3、内存拷贝，利用mmap()文件映射内存加速与内核空间的消息传递；即epoll使用mmap减少复制开销。

select、poll、epoll 区别总结：

1、支持一个进程所能打开的最大连接数

select	单个进程所能打开的最大连接数有FD_SETSIZE宏定义，其大小是32个整数的大小（在32位的机器上，大小就是32*32，同理64位机器上FD_SETSIZE为32*64），当然我们可以对进行修改，然后重新编译内核，但是性能可能会受到影响，这需要进一步的测试。
poll	poll本质上和select没有区别，但是它没有最大连接数的限制，原因是它是基于链表来存储的
epoll	虽然连接数有上限，但是很大，1G内存的机器上可以打开10万左右的连接，2G内存的机器可以打开20万左右的连接

2、FD剧增后带来的IO效率问题

select	因为每次调用时都会对连接进行线性遍历，所以随着FD的增加会造成遍历速度慢的“线性下降性能问题”。
poll	同上
epoll	因为epoll内核中实现是根据每个fd上的callback函数来实现的，只有活跃的socket才会主动调用callback，所以在活跃socket较少的情况下，使用epoll没有前面两者的线性下降的性能问题，但是所有socket都很活跃的情况下，可能会有性能问题。

3、消息传递方式

select	内核需要将消息传递到用户空间，都需要内核拷贝动作
poll	同上
epoll	epoll通过内核和用户空间共享一块内存来实现的。

总结：

综上，在选择select，poll，epoll时要根据具体的使用场合以及这三种方式的自身特点。

1、表面上看epoll的性能最好，但是在连接数少并且连接都十分活跃的情况下，select和poll的性能可能比epoll好，毕竟epoll的通知机制需要很多函数回调。

2、select低效是因为每次它都需要轮询。但低效也是相对的，视情况而定，也可通过良好的设计改善

上一篇 IP地址的三种表示格式及在Socket编程中的应用

下一篇 PHP中引用的详解(引用计数、写时拷贝)

顶

56

踩

2

主题推荐

异步

socket

i/o

linux内核

操作系统

猜你在找

- cs硕士妹子找工作经历阿里人搜等互联网
- 【精品课程】Windows Server 2012 DHCP Server 管理
- C++学习之深入理解虚函数—虚函数表解析
- 【精品课程】JavaScript for Qt Quick(QML)
- libpcap使用
- 【精品课程】C语言及程序设计初步
- 割绳子的作者你如此歧视无视鄙视中国人这是何苦呢
- 【精品课程】微信公众平台开发入门
- Linux下端口复用SO_REUSEADDR与SO_REUSEPORT
- 【精品课程】零基础学Java系列从入门到精通


准备好了么？跳吧！

更多职位尽在 CSDN JOB


unity3D模型师	我要跳槽	人工智能/机器学习/数据挖掘/数据科学,	我要跳槽
北京络捷斯特科技发展有限公司	8-16K/月	携程旅行网	10-20K/月
资深3D模型	我要跳槽	020客服专员	我要跳槽
成都动鱼数码科技有限公司	6-15K/月	广州星巴达信息技术有限公司	3-6K/月

查看评论


35楼 vbandplc 2015-04-23 21:21发表

 哈哈 人都好贪心。我也想要demo 快来满足我吧


34楼 jkand1 2015-02-28 22:45发表

 不错，很全面


33楼 ShirokilnCS 2015-01-05 11:16发表

 学习

32楼 luzi1024 2014-12-13 10:04发表

 学习，感谢分享

31楼 difoss 2014-12-04 13:04发表

 引用"zhouming0100"的评论:

关于阻塞的说话，略微有点不对，当前线程I/O阻塞，线程挂起，不会处理线程的其它消息，直到该线程解除阻...

同意楼上观点，麻烦作者写东西前仔细推敲一番。

30楼 difoss 2014-12-04 13:02发表

 引用"zhouming0100"的评论:

关于阻塞的说话，略微有点不对，当前线程I/O阻塞，线程挂起，不会处理线程的其它消息，直到该线程解除阻...

同意楼上观点，麻烦作者写东西前仔细推敲一番。

- 29楼 difoss 2014-12-04 12:59发表
- 还有后面跟着那句“通过select通知调用者”，这显然就有问题，究竟怎么个通知法？
- 应该写成：“通过返回值通知调用者本次调用时的IO情况。”
- 28楼 difoss 2014-12-04 12:51发表
- 引用“liaokailin”的评论：

同步IO和异步IO的区别就在于：数据拷贝的时候进程是否阻塞！

阻塞IO和非阻塞IO的区别就在于：应...
- 把主语的修饰部分和宾语严格对应起来，后面那句明显是反了。
- 27楼 一枪尽骚、魂 2014-11-01 14:53发表
- 赞一个
- 26楼 zhangkaijia403 2014-10-22 15:37发表
- 楼主，请问这些截图是出自哪本书的，看图就直接清晰了，谢谢！
- 25楼 asffaf 2014-10-07 10:36发表
- “例如，我们在socket中调用recv函数，如果缓冲区内没有数据，这个函数就会一直等待，直到有数据才返回。而此时，当前线程还会继续处理各种各样的消息。”
- 能不能不要误导人？recv 没返回，当前线程还会处理各种各样的消息？线程是挂起状态好吗！
- Re: 真实的归宿 2014-10-07 15:31发表
- 回复asffaf: 我只是读书笔记的总结。我也不是什么专家老师，没有误导任何人。
如果当前线程异步处理IO，当然还可以处理其他消息，如果是同步，就是当前线程是挂起。而recv函数本身来说是阻塞的。
- 24楼 yitian_1989 2014-09-30 10:16发表
- 楼主给力
- 23楼 u013322907 2014-09-19 11:36发表
- 学习了
- 22楼 weiwei2012start 2014-08-28 19:27发表
- 不错，谢谢分享。
- 21楼 huanggang982 2014-08-27 11:04发表
- 引用“ciscosh”的评论：

楼主你能再给力点吗？这篇文章分析的太到位了，学习了！
- 写的非常好！学习了！
- 20楼 菠萝猫咪 2014-08-20 20:41发表
- 非常好
- 19楼 菜鸟学编程二号 2014-08-05 16:31发表
- 分析得很好，解决了我的一些疑惑。
- 18楼 zwz19911991 2014-06-30 23:53发表
- 思维清晰！这图太赞了！
- 17楼 左言zy 2014-06-15 10:34发表
- 嗯，挺利害的。
- 16楼 zhouming0100 2014-05-28 16:11发表
- 关于阻塞的说话，略微有点不对，当前线程IO阻塞，线程挂起，不会处理线程的其它消息，直到该线程解除阻塞。
- 15楼 WanderingHeart 2014-05-27 15:21发表
- "异步则指主动请求数据后便可以继续处理其它任务,随后等待I/O,操作完毕的通知,这可以使进程在数据读写时也不阻塞。(等待"通知")",对于你的这句话，可以使进程在数据的读写时不阻塞还是不怎么理解。
- 14楼 人生一抹儿绿色 2014-05-23 23:08发表
- 讲的很深奥，学习了



13楼 li603572310 2014-05-07 12:09发表



学习了。非常感谢。



12楼 SoulSweet 2014-05-05 09:51发表

好文要顶！！



11楼 wowocsdn 2014-02-27 17:22发表

原来描述的不错，有些小示例代码就更好



10楼 yxc135 2014-01-09 20:40发表

写得太好了。



9楼 mollywml 2014-01-02 17:02发表

你解释的同步异步，阻塞非阻塞的区别，我可以这么理解么：
同步异步是宏观上的，整个进程是否等待，阻塞非阻塞是微观上的，某个系统调用的函数是否等待。
这样理解可以么？

Re: majianfei1023 2014-09-15 12:36发表



回复u012632043：我觉得可以这么说吧，系统调用分两步，等待数据和拷贝数据，阻塞非阻塞是指数据没就绪的时候是等待数据，还是立即返回，而同步异步是指拷贝数据的时候同步需要阻塞拷贝完毕才返回，异步就是立即返回，然后等待通知完成事件就可以了。

8楼 brk1985 2013-12-02 23:04发表



有人也许会把阻塞调用和同步调用等同起来，实际上他是不同的。对于同步调用来说，很多时候当前线程还是激活的，只是从逻辑上当前函数没有返回而已。例如，我们在socket中调用recv函数，如果缓冲区中没有数据，这个函数就会一直等待，直到有数据才返回。而此时，当前线程还会继续处理各种各样的消息。
=====

这个例子理解不了，recv函数一般不是在监听到读事件后，才被调用吗？怎么会缓冲区没有数据还在等待？不解。。。

7楼 廖凯林 2013-09-17 13:19发表



同步IO和异步IO的区别就在于：数据拷贝的时候进程是否阻塞！
阻塞IO和非阻塞IO的区别就在于：应用程序的调用是否立即返回！
这段话是不是反了呀？

Re: 太灰浪 2013-10-21 14:34发表



回复liaokailin：我也觉得这两句话反了

Re: 真实的归宿 2013-09-17 16:29发表



回复liaokailin：并没有反。实际就是这样的。

Re: 小敏纸 2014-09-15 10:27发表



回复hguisu：如果没有反的话，你之前说的都是错误的，前后矛盾

Re: majianfei1023 2014-09-15 12:39发表



回复lanxuezaipiao：没有矛盾吧，阻塞是指等待数据的阶段是不是立即返回，而同步是指拷贝数据的阶段，调用阻塞直到拷贝完毕，而异步则是不阻塞，拷贝完毕会通知你。

Re: 奋力向上游 2014-12-04 10:35发表



回复majianfei1023：同步和异步，应该是种更抽象的概念。比如一个web服务依赖于其它几种服务协同工作。也算是一种io，同步io就是当这些服务结果返回时，获得结果。这里没有用等这个词，还是可以用非阻塞调用。而异步的话，是结果来时调用回调，是通知机制。这里同步和异步的概念与阻塞和非阻塞的关系，没有扯上关系，如果有这层关系，是实际上是有系统调用时，用到的阻塞和非阻塞io。如果异步，人家非要用阻塞式的io，就是阻塞到结果来时，那就有了阻塞式的异步io，不过感觉好别扭。

Re: 真实的归宿 2013-09-17 16:29发表



回复liaokailin：实际就是这样的。

Re: ctqctq99 2013-10-08 17:05发表



引用“hguisu”的评论：
回复liaokailin：实际就是这样的。

他的意思可能是
阻塞IO和非阻塞IO的区别就在于：应用程序的调用是否立即返回！
这句话说反了。应该是非阻塞IO和阻塞IO的区别就在于：应用程序的调用是否立即返回！
不知道我的理解对不对？

6楼 [酥西黄](#) 2013-08-06 13:09发表



学习中

5楼 [a574490399](#) 2013-07-24 10:49发表



楼主给力、学习了。

4楼 [ccssddnnbbookkee](#) 2013-06-04 10:45发表



原来是说C语言

3楼 [honghuzhilangzixin](#) 2013-05-28 14:39发表



跟<http://www.ibm.com/developerworks/cn/linux/l-async/#N10164>讲的矩阵模型不大一样啊。尤其
“同步IO引起进程阻塞，直至IO操作完成。
异步IO不会引起进程阻塞。”

这句话

Re: [真实的归宿](#) 2013-05-28 15:13发表



回复[honghuzhilangzixin](#): "同步阻塞， 同步非阻塞， 异步阻塞， 异步非阻塞"
如果不理解的话肯定把人搞晕了。
其实你先理解同步/异步， 阻塞/非阻塞的区别。
<http://www.ibm.com/developerworks/cn/linux/l-async/> 这篇文章曾经引起很大的热议。

2楼 [ciscohsh](#) 2013-04-22 16:33发表



引用“[ciscohsh](#)”的评论：
楼主你能再给力点吗？这篇文章分析的太到位了，学习了！

1楼 [ciscohsh](#) 2013-04-22 16:33发表



楼主你能再给力点吗？这篇文章分析的太到位了，学习了！

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题	Hadoop	AWS	移动游戏	Java	Android	iOS	Swift	智能硬件	Docker	OpenStack
VPN	Spark	ERP	IE10	Eclipse	CRM	JavaScript	数据库	Ubuntu	NFC	WAP
jQuery	BI	HTML5	Spring	Apache	.NET	API	HTML	SDK	IIS	Fedora
XML	LBS	Unity	Splashtop	UML	components	Windows Mobile	Rails	QEMU	KDE	Cassandra
CloudStack	FTC	coremail	OPhone	CouchBase	云计算	iOS6	Rackspace	Web App	SpringSide	Maemo
Compuware	大数据	aptech	Perl	Tornado	Ruby	Hibernate	ThinkPHP	HBase	Pure	Solr
Angular	Cloud Foundry	Redis	Scala	Django	Bootstrap					

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持
京 ICP 证 070598 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved