第34卷 第23期 计 算 机 工 程 2008年12月
Vol.34 No.23 Computer Engineering December 2008
·网络与通信· 文章编号：1000—3428(2008)23—0142—02 文献标识码：A 中图分类号：TP393.02

Linux系统中网络I/O性能改进方法的研究

李 涛，房鼎益，陈晓江，冯 健
(西北大学信息学院，西安 710127)

摘 要：选择并设计高效的网络I/O模型是改善服务器性能的关键。该文通过对Linux系统中几种网络I/O模型的分析和研究，提出3种改善网络I/O性能的方法，并讨论这3种方法在Linux系统中的实现技术。实验结果验证了该方案的有效性。
关键词：网络I/O；Linux系统；Web服务器；性能改善

Research on Improvement of Network I/O Performance in Linux

LI Tao, FANG Ding-yi, CHEN Xiao-jiang, FENG Jian
(School of Information, Northwest University, Xi'an 710127)

【Abstract】 To choose and design effective network I/O model is the key point to improve the performance of Web server. By analyzing several network I/O models in Lunix system, three methods to improve the performance of these network I/O models are proposed. How to implement these methods is also discussed. Experimental results show its efficiency.
【Key words】 network I/O; Linux system; Web server; performance improvement

1 概述

网络硬件的发展为网速的提高提供了很好的硬件平台，同时，软件方面不断设计出各种网络I/O模型，以最大限度地使用现有硬件资源，提高网络性能。由于Linux系统中的每种网络I/O模型都是针对新的网络环境提出的，因此在设计网络程序时要根据不同网络环境选择合适的网络I/O。

本文针对Linux系统中各种网络I/O的原理和性能进行分析，给出每种模型适用的网络环境。在实际网络环境中，服务器端要接收成千上万的并发连接，如何高效地响应和管理这些连接已成为衡量服务器性能的关键因素。由于网络传输的某些原因，有些连接的数据传输量可能很小，甚至没有，若这种连接增多的话，会对服务器性能产生较大影响。

本文对Linux系统中的各种网络I/O进行分析，提出高并发网络I/O模型的一般方法，并通过实验数据验证这些方法的正确性。

2 Linux系统网络I/O原理[1]

2.1 阻塞模式(Blocking I/O)

阻塞模式是常见的、最简单的网络I/O模式，应用程序发送一个I/O请求，该I/O操作不能立刻完成，比如，在读网络接口数据时，网络接口缓冲区还没有数据到达或数据没有完全接收；写操作时网络接口缓冲区已满或空闲空间小于要写入的数据量。这2种情况会导致进程阻塞，直到网络接口缓冲区有数据到达或有足够空间发送数据，系统才会唤醒阻塞进程，完成I/O操作。阻塞I/O如图1所示。

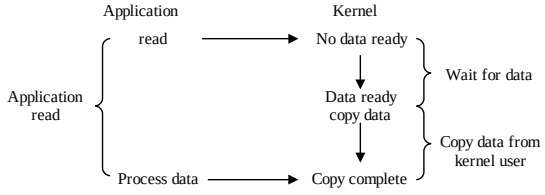

图1 阻塞I/O

2.2 非阻塞模式(Nonblocking I/O)

应用程序向内核发送一个I/O请求，若网络接口没有可供操作的数据，系统向用户返回一个错误，让用户在将来合适的时候再请求I/O操作，从而避免进程阻塞，这就是非阻塞模式，如图2所示。

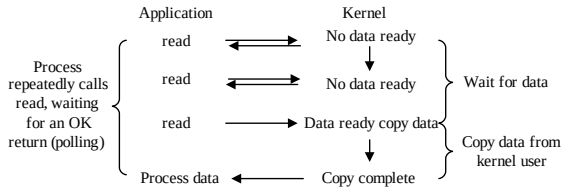

图2 非阻塞I/O

2.3 多路复用I/O(I/O Multiplexing)

虽然非阻塞I/O可以避免进程进入阻塞状态，但并不知道何时可以进行I/O操作，只能通过不断地发送I/O请求进行测试，直到能成功地完成I/O操作，这种轮询操作会浪费不少系统资源。

多路复用I/O提供了一个解决方法。使用select/poll操作，进程可在多个socket上等待网络事件，当其中某个socket发生某个网络事件时，用户可通过查看网络事件对该socket进行I/O操作。但当所有socket都没有网络事件发生时，进程还会阻塞起来。所以，多路复用I/O模型本质上还是基于阻塞I/O的。

在该模型中，应用程序要不断地向内核写入socket描述

基金项目：陕西省国际科技合作基金资助重点项目(2006KW-21)；陕西省教育厅产业化基金资助重点项目(05JC27)
作者简介：李 涛(1982－)，男，硕士研究生，主研方向：计算机网络，分布式系统；房鼎益，教授、博士生导师；陈晓江，讲师、博士研究生；冯 健，博士研究生
收稿日期：2008-05-20 **E-mail：**tohy2005@163.com

符，导致过多的用户空间和内核空间间的数据拷贝，且内核还要不断扫描 socket 集来检测 socket 状态，若多数 socket 没有发生事件，就会浪费系统资源，该模式如图 3 所示。

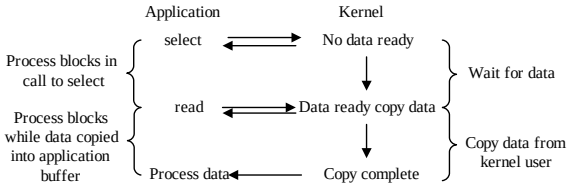

图 3　多路复用 I/O

2.4　信号驱动 I/O

信号驱动 I/O 可实现进程 I/O 操作无阻塞地进行。当可以进行 I/O 操作时，系统异步通知进程，而无需进程阻塞等待 I/O 操作的完成或网络事件的发生。这比前面几种 I/O 模型更具灵活性。但在多进程服务器中，信号驱动 I/O 存在一个问题，即信号在产生和传递到目的进程之间，状态标志可能发生改变，如图 4 所示。

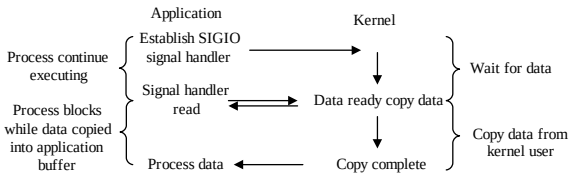

图 4　信号驱动 I/O

2.5　异步 I/O(Asynchronous I/O)

异步 I/O 是 POSIX 规范的一部分。当系统完成 I/O 操作时，系统通知进程操作结果。这种机制容易和信号驱动 I/O 混淆，两者的区别是：异步 I/O 返回时，I/O 操作已经完成，返回的是 I/O 操作的结果；信号驱动 I/O 只是通知进程可以开始进行 I/O 操作，进程得到这个信号后才开始 I/O 操作，如图 5 所示。

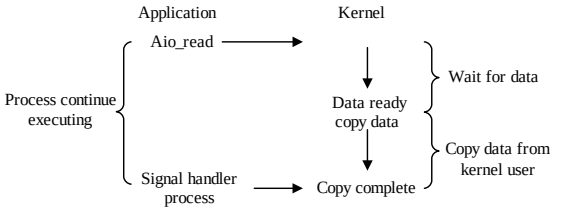

图 5　异步 I/O

3　Linux 网络 I/O 性能分析

3.1　各个网络 I/O 模型性能分析比较

阻塞模型较为直观，服务器端为每个请求的客户连接建立一个线程或者进程，并处理这个连接数据。若有大量客户请求连接，系统就建立大量线程或者进程，线程间的上下文切换使系统性能大受影响。这种模型适合并发数不多或服务器负载不大的情况。

非阻塞 I/O 主要应用在单进程服务器中。若服务器希望在当前请求的 I/O 未完成时去处理其他事务，就要选择这种模型，而该模型并不知道何时再次进行 I/O 请求操作，虽然可通过轮询的方法查询连接状态，但轮询操作会造成 CPU 的浪费。多路复用针对上述问题提出一种解决办法，进程可在多个描述符上等待网络 I/O 事件(可读、可写)，担当没有任何网络事件发生时，进程会进入阻塞状态。因此，该模型适合多并发连接的情况，且这些并发连接大多要处于活跃状态。

信号驱动 I/O 的本质属于异步 I/O，但这种模型存在缺点，即传统的 UNIX/Linux 信号是不排队的，对于某个进程，当一个信号处于挂起状态时，若同一个信号再次产生则会被丢弃。这就不能保证每个事件的 SIGIO 信号都被递送到进程。所以，当有 SIGIO 到达时进程并不能确定有多少事件发生了，而要检查所有描述符；另外，SIGIO 只告诉进程有 I/O 事件发生，并没有关于事件是发生在哪些描述符上的信息，所以，检查也是必须的。这种检查通常是调用 select/poll 来完成的，因此，一般服务器端很少使用这种模型。

Linux 以 C 函数库形式提供 AIO 操作函数，而 C 库通过建立用户级线程实现 AIO。如果应用程序请求大量异步 I/O，就会产生大量线程，这些线程间的切换会导致系统性能下降。因此，AIO 适合并发数不多的服务器。

epoll模型[2]针对的是服务器高并发的情况。它的I/O效率不会随连接数的增加而下降，对一个数目较大的socket集合来说，并不一定每个socket在任何时候都处于活跃状态，可能只有部分socket处于活跃状态，epoll模型并不像传统select/poll的轮询方法那样扫描整个socket集合，epoll只给用户返回活跃的socket。另外，epoll利用mmap让内核和用户空间共用一块内存，省去了内核和用户空间间数据的拷贝，提高系统效率。这种模型适用于高并发连接服务器且活跃的连接不是很高的情况。若大部分连接都不处于活跃状态，那么这种模型的效率不一定比select/poll高。

3.2　结果分析

从以上对 Linux 网络 I/O 性能的分析可以看出，不同环境要选择不同的网络 I/O 模型，这样才能设计出较好的服务器程序。

4　提高网络 I/O 效率的方法

本文在高并发的网络环境下，提出以下 3 种高效网络 I/O 的设计方法。

4.1　高效的异步通知

因为阻塞 I/O 模型不存在异步通知，所以该模型的吞吐率较低；多路复用模型(select/poll 模型)可在一个进程中管理一个 socket 集合，但该模型实质上也属于阻塞模型，当集合中没有 socket 发生网络事件时进程就阻塞，内核也是通过轮询 socket 集合得到网络事件，并不是操作系统通知给用户程序，所以，当 socket 数量增大时，系统性能会线性下降；AIO 系统有较大改善，但该模型的异步不是真正的异步，AIO 只和 I/O 相关，并不能关联 socket，当 I/O 处理完成后，并不能知道和哪个事件相关的 I/O 完成了，这个就产生了线程切换问题；epoll 模型通过 callback 方法实现系统异步通知，socket 集合中活跃的 socket 通过调用 callback 函数通知客户程序，省去了轮询时间，提高了网络性能。

4.2　热连接和冷连接

在高并发连接的情况下，由于网络或其他原因，并非每个连接的 socket 都在活跃状态，即存在数据的发送和接收，因此将系统资源集中应用在活跃的 socket 上，不仅节省了系统资源，而且还提高了服务器的处理效率。select/poll 模型没有区分热连接和冷连接，内核每次都要轮询集合中所有 socket，浪费了系统资源。epoll 模型对 select/poll 进行改进，内核只针对活跃的 socket 进行处理。

4.3　内核和用户空间之间消息传递

当系统内核完成 I/O 处理后需向用户空间发送通知消

（下转第 146 页）