第 35 卷 第 3 期 计 算 机 工 程 2009 年 2 月
Vol.35 No.3 Computer Engineering February 2009
·网络与通信· 文章编号：1000—3428(2009)03—0103—03 文献标识码：A 中图分类号：TP393

高性能服务器底层网络通信模块的设计方法

王文武[1,2]，赵卫东[1,2]，王志成[1,2]，陈 悦[1,2]，韩下林[1,2]
(1. 企业数字化技术教育部工程研究中心，上海 200092；2. 同济大学 CAD 研究中心，上海 201804)

摘 要：在对 I/O 完成端口进行底层封装的基础上，提出一种具有高性能的、可扩展性的通用网络通信模块设计方案。该方案采用多种系统性能优化技术，如线程池、对象池和环形缓存区等。该模块在 Win32 平台上用 C++开发完成，经过严格的压力和性能测试后，实验结果表明该模块能够支持海量并发连接，具有较高的数据吞吐量，在实际项目应用中也取得了良好的表现。

关键词：完成端口；服务器；多线程；线程池；对象池；缓存区

Design Method of Underlying Module of Network Communication for High Performance Server

WANG Wen-wu[1,2], ZHAO Wei-dong[1,2], WANG Zhi-cheng[1,2], CHEN Yue[1,2], HAN Xia-lin[1,2]
(1. State Education Commission Engineering Center for Enterprise Digital Technology, Shanghai 200092;
2. Research Center of CAD, Tongji University, Shanghai 201804)

【Abstract】On the base of underlying encapsulation for I/O Completion Port(IOCP), this paper presents a design solution with high performance and scalable module of generic network communication, which emploies a variety of optimization techniques of system performance, such as thread pool, object pool and ring buffer. The module is developed in C++ programming language on Win32 platform. Experimental results show that the module can support massive concurrent connections, and has higher data throughput based on severe pressure and performance tests. The proposed solution has also got a good performance in the actual project application.

【Key words】completion port; server; multi-threading; thread pool; object pool; ring buffer

1 概述

要设计与开发出一款高性能的服务器(如网游服务器、Web 服务器和代理服务器等)，一般都采用高效率的网络 I/O 模型[1]。Linux 平台上经常会采用 epoll 模型，而在 Win32 平台上完成端口(以下简称 IOCP)模型[2]是设计与开发高性能的、具有可伸缩性的服务器的最佳选择，它可以支持海量并发客户端请求。多线程编程是服务器端开发常用技术，多线程必然涉及线程间的通信与同步。如果使用不当，也会影响到系统的性能，必须谨慎设计才能保证系统良好运行。减少数据拷贝以及小对象频繁创建与销毁是一种很重要的提高系统性能手段，可通过设计内存池或对象池加以解决。这几个问题的提出，说明实际的高性能服务器研发比较复杂，尤其是采用高效 I/O 模型来架构服务器时，更是增加了开发的难度，原因是这些模型的机制比较复杂。

底层网络通信模块是服务器应用程序的核心模块，也是高性能服务器的最基础模块之一。它主要的功能是接收海量并发连接、接收网络数据包、暂存和发送应用逻辑层的逻辑数据包，所以，它也是上次应用逻辑和底层网络之间通信的媒介。

2 IOCP 机制

要实现一个并发的网络服务器，比较简单的模型是：每当一个请求到达就创建一个新线程，然后在新线程中为请求服务。这种模型减轻了实际开发的复杂度，在并发连接较少的情况下可以考虑使用，然而在高并发需求下并不适用。高并发环境中，创建和销毁大量线程所花费的时间和消耗的系统资源是巨大的，而且会加重线程调度的负担，同时线程上下文切换(context switch)也会浪费许多宝贵的 CPU 时间。

为了提高系统性能，首先必须有足够的可运行线程来充分利用 CPU 资源，但线程的数量不能太多。事实上，具体工作线程的数量和并发连接数量不是直接相关联的。在 Win32 平台下开发高效的服务器端应用程序，最理想的模型是 IOCP 模型，该模型解决了一系列系统性能瓶颈问题。

IOCP 提供了最好的可伸缩性，而且其执行效率比较高，采用这种网络模型可能会加大开发的复杂度，但却是 Windows 平台上唯一适用于开发高负载服务器的技术。IOCPWindows 系统的一种内核对象[3]，也是 Win32 下最复杂的一种 I/O 模型，它通过一定数量的工作线程对重叠 I/O 请求进行处理，以便为已经完成的 I/O 请求提供服务，相对其他 I/O 模型，它可以管理任意数量套接字句柄。它主要由等待线程队列和 I/O 完成队列 2 个部分组成。一个完成端口对象可以和多个套接字句柄相关联，当针对某个套接字句柄发起的异步 I/O 操作完成时，系统向该完成端口的 I/O 完成队列加入一个 I/O 完成包。于此同时，工作线程调用 GetQueuedCompletionStatus(以下简称 GQCS)时，如果 I/O 完

基金项目：广东省教育部产学研结合基金资助项目“基于 RFID 技术石化产品计量监控管理系统应用研究与开发”(2007A090302094)
作者简介：王文武(1983－)，男，硕士研究生，主研方向：CAD，企业信息化；赵卫东，研究员、博士生导师；王志成，博士；陈 悦、韩下林，硕士研究生
收稿日期：2008-11-30 **E-mail：**jerry.wenwu@gmail.com

成队列中有完成包，当前调用就会返回，取得数据进行后续的处理。

成功创建一个完成端口后，便可开始将套接字句柄与对象关联到一起。但在关联套接字之前，首先必须创建一个或多个工作者线程为完成端口提供服务。

3 模块设计方案

3.1 架构设计

在充分考虑服务器性能和扩展性的基础上，本文提出了基于三层结构的系统设计方案，该模块的架构如图 1 所示。

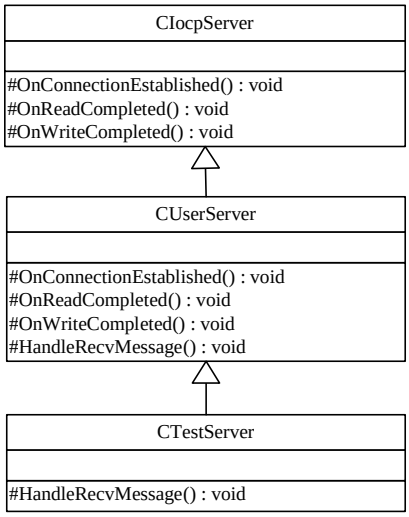

图 1 模块设计架构

图 1 看上去并不复杂，但却是一个兼顾可扩展性和高性能的架构，在实际项目应用中也取得了很好的表现，下面是对图 1 主要类的功能分析。

CIocpServer 类是完成端口服务器基本通信类，它使用 Windows 平台特有的 IOCP 机制，对网络通信模型进行底层封装。提供了基本的服务器端网络通信功能，这些功能主要有开启服务器、关闭服务器、管理客户端连接列表、管理未决的接受请求列表、发出异步操作等。同时通过多态机制向它的派生类提供以下基本扩展接口：

(1)新连接确立的处理接口。

(2)客户端断开连接时的处理接口。

(3)连接出现错误时的处理接口。

(4)从客户端接收完数据后的处理接口。

(5)向客户端发送完数据后的处理接口。

(6)拼包处理接口。

CUserServer 类继承 CIocpServer，在 CIocpServer 的基础上，CUserServer 加入了一些服务器逻辑处理功能，并且封装了 3 类数据队列和 3 类处理线程，分别如下：

(1)接收数据包队列及接收线程：接收队列用于存放接收到的数据包，此数据包还没有进行逻辑意义上的拼包，接收线程从此队列中取出数据包，并将其拼装成逻辑意义上完整的数据包加入到逻辑数据包队列中。

(2)逻辑数据包队列及逻辑处理线程：逻辑队列用于存放已经拼包成了逻辑意义上的数据包，逻辑处理线程对此类数据包进行逻辑解析，这里就是服务器的主要逻辑部分，有的数据包在处理完成后，可能是需要向客户端返回处理结果，此时就需要逻辑线程将处理完成的数据包放入发送数据包队列中。

(3)发送数据包队列及发送线程：发送队列存放待发送的数据包，发送线程根据数据包里的客户端套接字发送给特定客户端。

CTestServer 类是一个测试类，主要用于演示如何在 CUserServer 的基础上派生一个真正的应用服务器，并用于说明它需要重载实现 CUserServer 的哪些重要虚函数。

3.2 资源管理

用 IOCP 开发服务器时，当 I/O 发生错误时需要有效地释放与套接字相关的缓存区，如果对同一缓存区释放多次，就会导致内存释放的错误。当投递的异步 I/O 请求返回了非 WSA_IO_PENDING 错误时，要对此错误进行处理，通常执行 2 步操作：释放此次操作使用的缓冲区数据；关闭当前操作所使用的套接字句柄。同时 GQCS 调用会返回 FALSE，也要做上面 2 步相同的操作，这样就可能产生对同一缓存区进行重复释放的错误。解决的办法可以有 2 种：

(1)通过引用计数机制控制缓存区释放；

(2)使缓存区释放操作线性化。

该系统的设计采用了第(2)种解决方案，所谓的释放操作线性化是指把可能引起 2 次释放同一缓存区的操作合并为一次释放。如果在执行异步 I/O 操作过程中发生了非 WSA_ IO_ PENDING 错误，可以让 GQCS 返回时得知这个错误和发生错误时的缓存区指针，而不对该错误进行处理。通知的方式是，使用 PostQueuedCompletionStatus(以下简称 Post)函数抛出一个特殊标志的消息，这个特殊标志可以通过 GQCS 函数的第 2 个参数，即传送字节数来表示，可以选择任何一个不可能出现的值，比如一个负数。当然，如果通过单句柄数据或单 I/O 数据来传递也是可以的。而发生错误时的缓冲区指针，必须要通过单句柄数据或单 I/O 数据来传递。

把释放操作全放在 GQCS 函数里以后，对释放操作的处理就比较统一了。当然，为了实现真正的线性化和原子化，在释放操作的执行逻辑上需要对释放代码加锁以实现线程互斥(多线程情况下)。

3.3 包的乱序解决方案

如果在同一个套接字上一次提交多个异步 I/O 请求，肯定会按照它们提交的次序完成，但在多线程环境下，完成包处理次序可能和提交次序不一致。该问题的一个简单的解决方法是一次只投递一个异步 I/O 请求，当工作线程处理完该请求的完成数据包后，再投递下一个异步 I/O 请求。但这样做会降低服务器的处理性能。为了保证完成包处理次序和提交次序相一致，可以为每个连接上投递的请求都分配一个序号，单句柄数据中记录当前需要读取的单 I/O 数据的序号，如果工作线程获得的单 I/O 数据的序号与单句柄数据中记录的序号一致的话，就处理该数据。如果不相等，则把这个单 I/O 数据保存到该连接的 pOutOfOrderReads 列表中。

4 性能优化

在网络服务器的开发过程中，池(Pool)技术已经被广泛应用。使用池技术在一定层度上可以明显优化服务器应用程序的性能，提高程序执行效率和降低系统资源开销。这里所说的池是一种广义上的池，比如数据库连接池、线程池、内存池、对象池等。其中，对象池可以看成保存对象的容器，在进程初始化时创建一定数量的对象，需要时直接从池中取出一个空闲对象，用完后并不直接释放掉对象，而是再放到对象池中以方便下一次对象请求可以直接复用。其他几种池的设计思想也是如此，池技术的优势是，可以消除对象创建所

带来的延迟，从而提高系统的性能。

4.1 线程池

线程池[4]是提高服务器程序性能的一种很好技术，在Win32 平台下开发的网络服务器程序使用的线程池可分为两类：一类是由完成端口对象负责维护的工作线程池，主要负责网络层相关处理(比如投递异步读或写操作等)；另一类是负责逻辑处理的线程池，它是专门提供给应用层来使用的。本文提出了一种逻辑线程池的设计方案，线程池框架结构主要分为以下几个部分：

(1)线程池管理器：用于创建并管理线程，往任务队列添加数据包等，并可以动态增加工作线程。

(2)工作线程：线程池中的线程，执行实际的逻辑处理。

(3)任务接口：每个任务必须实现的接口，以供工作线程调度任务使用。

(4)任务队列：提供一种缓存机制，用于存放从网络层接收的数据包。

该通信模块使用了上述线程池的设计方案，从测试结果来看，当并发连接数很大时，线程池对服务器的性能改善是显著的。

该设计方案有个很好的特性，就是可以创建工作线程数量固定的线程池，也可以创建动态线程池。如果有大量的客户要求服务器为其服务，但由于线程池的工作线程是有限的话，服务器只能为部分客户端服务，客户端提交的任务只能在任务队列中等待处理。动态改变的工作线程数目的线程池，可以以适应突发性的请求。一旦请求变少了将逐步减少线程池中工作线程的数目。当然线程增加可以采用一种超前方式，即批量增加一批工作线程，而不是来一个请求才建立创建一个线程。批量创建是更加有效的方式，而且该方案还限制了线程池中工作线程数目的上限和下限，确保线程池技术能提高系统整体性能。

4.2 对象池

对象池[5-6]是针对特定应用程序而设计的内存管理方式，在某种场合下内存的分配和释放性能会大大提升。默认的内存管理函数(new/delete 或 malloc/free)有其不足之处，如果应用程序频繁地在堆上分配和释放内存，那么就会导致性能损失，并且会使系统中出现大量的内存碎片，降低内存的利用率。

所谓对象池就是应用程序可以通过系统的内存分配调用预先一次性申请适当大小的内存块，然后可以根据特定对象的大小，把该块内存分割成一个个大小相同的对象。如果对象池中没有空闲对象使用时，可以再向系统申请同样大小的内存块。如果对象使用完毕后直接放到对象池中，这种内存管理策略能有效地提升程序性能。

4.2.1 对象池的应用

当服务器接受一个客户端请求后，会创建成功返回一个客户端套接字句柄。如果出现大量并发客户端连接请求时，就会出现频繁地分配和释放对象的情况，这个过程可能会消耗大量的系统资源，有损系统性能。WinSock2 还提供一个接受扩展函数 AcceptEx，它允许在接受连接之前就事先创建一个套接字句柄，使之与接受连接相关联。在调用 AcceptEx 时，可以直接把该句柄作为参数传递给 AcceptEx。有了这个保证，可以通过采用对象池技术来提升系统性能，可以在接受连接之前就创建一定数量的套接字句柄，随着新连接请求的到来将句柄分配出去，当客户端断开连接后，把相应句柄重新放入套接字对象池中。

另外需要用到对象池的地方是，在每一次投递 WSASend 或 WSARecv 操作时，都要传进一个重叠结构体参数。可以提前创建一个重叠结构体对象池，当发起异步 I/O 操作时，先从池中取一个结构体对象，用完之后并不直接销毁，而是再放回对象池以便以后重复利用。创建的结构体数量取决于完成端口的处理效率，如果处理效率比较高，则数量可能就少些，反之，就需要多创建些对象。

该系统所设计的对象池是线程安全的，可以被多个线程共享，在获得和释放对象时都需要加锁，从而保证线程间互斥访问对象池。

4.2.2 对象池的优点

与系统直接管理内存相比，对象池在系统性能优化方面主要有如下优点：

(1)针对特殊情况，例如需要频繁分配和释放固定大小的对象时，不需要复杂的分配算法和线程同步。也不需要维护内存空闲表的额外开销，从而获得较好的性能。

(2)由于直接分配一定数量的连续内存空间作为内存块，因此一定程度上提高了程序局部性能，提升了应用程序整体性能。

(3)比较容易控制页边界对齐和内存字节对齐，基本没有内存碎片问题。

4.3 环形缓存区

基于 TCP 协议的服务器应用程序，拼包处理过程必不可少。由于要从接收缓存中分解出一个个逻辑数据包，因此一般都要涉及内存拷贝操作，过多的内存拷贝必然降低系统性能。

当然，就逻辑数据包的拼装问题而言，也完全可以避免数据拷贝操作，方法是使用环形缓冲区。本文所说的环形缓冲区是具体这种特征的接收缓冲区，在服务器的接收事件里，当处理完了一次从缓冲区里取走所有完整逻辑包的操作后，可能会在缓冲区里遗留下来新的不完整数据包。使用了环形缓冲区后，就可以不将数据重新复制到缓冲区首部以等待后续数据的拼装，可以根据记录下的队列首部和队列尾部指针进行下一次的拼包操作。

环形缓冲区在 IOCP 的处理中，甚至在其他需要高效率处理数据收发的网络模型的接收事件处理中，是一种被广泛采用的优化方案。

5 实验结果

为了证明论文中系统优化的方法能获得预期的性能优势，对内存池和系统整体性能进行了实验测试。测试硬件是：CPU：AMD Turion 64，内存：1 024 MB，网络：100 MB 局域网，操作系统：Windows XP Professional SP2。

测试 1：对象池性能测试

由表 1 可以看出，由于使用对象池来分配小对象的内存，速度提高了 52.48%，使得内存分配获得了显著的效率提升。速度提高的原因可以归结为以下几点：

(1)除了偶尔的内存申请和销毁会导致从进程堆中分配和销毁内存块外，绝大多数的内存申请和销毁都由对象池在已经申请到的内存块中进行，而没有直接与进程打交道，而直接与进程打交道是很耗时的操作。

(2)这是在单线程环境的对象池，在多线程环境下，由于加锁，因此速度提高的会少些。

(下转第 114 页)