Journal of Computer Applications
计算机应用,2012,32(3):812-815
ISSN 1001-9081
CODEN JYIIDU
2012-03-01
http://www.joca.cn

文章编号:1001-9081(2012)03-0812-04
doi:10.3724/SP.J.1087.2012.00812

完成端口实现高性能服务端通信层的关键问题

廖宏建*,杨玉宝,唐连章
(广州大学 网络与现代教育技术中心,广州 510006)
(*通信作者电子邮箱 soluloo@126.com)

摘 要:为实现高性能稳定的网络通信服务,对完成端口(IOCP)应用中信息识别与提取、资源管理、消息乱序处理3个关键问题进行了分析,提出了IOCP参数扩展、内存池、对象池、环形缓冲等改进的解决方法。使用这些方法对IOCP底层进行了封装,并设计和实现了面向企业应用的可扩展网络程序通信模块。压力和性能测试结果表明,该模块能在合理资源消耗基础上支持海量并发连接,具有较高的数据吞吐量,在实际项目应用中也表现出了良好的性能。

关键词:完成端口;I/O模型;套接字;内存池;环形缓存
中图分类号:TP393 **文献标志码**:A

Key issues of high-performance server communication layer by using I/O completion port

LIAO Hong-jian*, YANG Yu-bao, TANG Lian-zhang
(*Network and Modern Education Technology Center, Guangzhou University, Guangzhou Guangdong* 510006, *China*)

Abstract: In order to achieve high-performance and stable network communication service, the key issues of client information identifying and extracting, resource management and message sequence dealing in I/O Complete Port (IOCP) development were analyzed. And the improved methods of IOCP parameter extension, memory pool, object pool and ring-buffer were proposed respectively. On the basis of underlying encapsulation for IOCP using these methods, a scalable network communication module for enterprise applications was designed and implemented. The experimental results show that the module can support massive concurrent connections, and has higher data throughput by reasonable resource consumption. The proposed solution has also showed good performance in the actual project application.

Key words: I/O Complete Port (IOCP); I/O model; socket; memory pool; ring buffer

0 引言

网络服务程序通信层的"高性能"主要表现为对客户端更大的响应规模、更高的并发处理能力、更高的数据吐吞量、更低的系统资源消耗等方面[1],这对服务端通信层采用的技术及设计提出了更高的要求。为此,微软研究数年提出了完成端口(I/O Complete Port, IOCP)机制,它把重叠I/O操作完成的事件通知放入系统维护的一个队列,并唤醒某个工作线程来处理相应的消息。重叠操作是在与IOCP相关联的一个或多个文件句柄上进行的,大大减少了线程上下文切换,最大限度地提高了系统并发量[2],能满足"高性能"要求。

IOCP是一个高效但复杂精巧的内核对象,微软的SDK只提供了简单的说明文档和示例代码[3],现有的相关文献或以描述IOCP机制的使用方法和步骤为主[4-5],或局限于IOCP原理探讨[6],或出于应用项目商业版权的考虑,局限于系统的模块设计[7],对构建大响应规模服务程序时遇到的棘手问题给出全面具体解决方法的较少。在IOCP开发中,存在的典型问题有:客户端信息识取、资源管理与访问紊乱、消息乱序,这3个问题的有效解决是构建稳健、高效IOCP服务程序的关键。在研究现有资料并结合开发实践基础上,下面讨论这些问题起因并给出解决办法。

1 IOCP关键问题分析

1.1 客户端信息的识别与提取

在IOCP应用中必定要处理大量客户端的I/O请求,当大量客户端连接服务器并且一个异步I/O操作成功完成时,服务线程对操作结果进行处理就必须要知道这个I/O是来自哪个客户端,以及I/O数据存在何处(如该Socket所对应的客户端Sockaddr_in结构体数据、存放网络数据的缓冲区等)。

文献[8]给出了Map查找的方法,即创建一个与该socket一一对应的客户端底层通信对象并将对象放入一个类似于list或map的容器中,需要使用时使用容器的查找算法根据socket值找到它所对应的对象信息。在客户端连接数不多的情况下这种方法简单可行,但是当客户端连接量很大时,频繁查表非常影响效率。

其实在IOCP中,工作者线程调用API函数GetQueuedCompletionStatus取得一个I/O完成时,会捎带两个具有可扩展的参数,一个是ULONG_PTR类型的"CompletionKey(完成键)";另一个是OVERLAPPED结构的指针。其中"完成键"是一个套接字句柄首次与IOCP对象关联时所使用的一个参数,既然在I/O完成后可取回这个数据,那么可以把它当一个指针使用,在关联之前对它指向的数据结构进行扩展,设计为一个包含客户端信息的结构,在完成返

收稿日期:2011-09-18;修回日期:2011-11-16。

作者简介:廖宏建(1980-),男,湖南衡阳人,讲师,硕士,主要研究方向:计算机网络; 杨玉宝(1979-),男,安徽庐江人,讲师,硕士,主要研究方向:教育信息化; 唐连章(1964-),男,湖南益阳人,副教授,主要研究方向:高性能计算、分布式计算。

回时,应用层通过读取完成键指向的这个数据结构来识别客户端。

IOCP 利用 OVERLAPPED 结构进行重叠 I/O 操作,同时规定,如果要投递一个重叠 I/O 操作,必须要带有 OVERLAPPED 结构,即要把指向一个 OVERLAPPED 结构的指针包括在其参数中。如上所述,在操作完成后也可以拿回这个指针,但是单凭这个指针所指向的 OVERLAPPED 结构,应用程序并不能分辨完成的是哪个类型的操作。不过基于上面的认识,只需要自定义一个扩展的 OVERLAPPED 结构,在其中加入所需的跟踪信息,当操作结束并取得指向 OVERLAPPED 结构的指针后,可以用 CONTAINING_RECORD 宏取出指向扩展结构的基址,进而读取里面的跟踪信息来判断操作类型。

同时,I/O 操作时系统将 OVERLAPPED 结构的地址传给核心,核心完成相应的操作后仍然通过这个结构传递操作结果,这样 OVERLAPPED 结构中已经包含了数据位移参数,扩展的结构中只要包含数据缓存地址及缓存大小即可得到网络数据。

可见,这两个参数提供具有很强的"伸缩"作用,巧妙和充分利用这两个参数,就能解决客户端信息识别与数据提取的问题。

1.2　系统内存资源管理

内存管理是使用好 IOCP 的核心问题。使用 IOCP 机制异步处理大量请求时,很容易出现因内存不足、访问紊乱致使程序崩溃等问题。这不是 IOCP 的特有问题,而是应用中引起的系统资源耗尽问题。为提高资源使用效率,必须对资源管理进行合理的设计。

1)异步 I/O 缓冲区的分配与利用。

在 Winsock 中,核心模式驱动程序(Kernel Model Driver, KMD)负责缓冲区管理,由 AFD. SYS 文件实现。当调用 WSASend( )函数异步操作时,AFD. SYS 将把数据拷贝到内部 AFD 缓冲区,然后 WSASend 函数立即返回,由 AFS. SYS 在后台负责把数据发送出去。从客户端接收数据的情况相似。在这个过程中会出现调用 WSASend 发送错误的情况,但错误码是 WSA_IO_PENDING,这表示 AFD 缓冲区已满而无法拷贝程序缓冲区数据。这时系统会将程序缓冲区加载到页面内存并锁定,直到 AFD 缓冲区有空间来拷走程序缓冲区数据,并给 IOCP 一个完成消息,调用返回时解锁页面内存。当大量客户端连接并投递大量异步重叠 I/O 时,就会出现大量未决的 I/O 缓存被加载到内存并加锁。同时,系统会锁定包含缓冲区的整个内存页面。当服务程序锁定的内存达到系统规定的内存分页极限时,就会产生 WSAENOBUFS 错误。

文献[9]给出了 3 点注意事项,除了强调使用工作线程执行异步操作这一点外,其余无甚裨益。解决这个问题的思路是减少可能被锁定的缓存区,目前可行的一个方法是投递零字节缓冲区来实现。先投递零字节空缓冲区的异步读取操作 AZeroByteRead( ),当操作返回时表示有网络数据可以读取,再投递非零字节缓冲区的 WSARecv 函数来进行真实数据的读取。在真实数据读取完成返回时再投递 AZeroByteRead( ),如此循环。这样就总能对被锁定的内存进行解锁。此外,尽量使用 VirtualAlloc 函数分配系统页面大小倍数的读写缓存区,这样分配的缓冲块才是与页面对齐的,这样就能减少总的内存锁容量。

以上只是内存使用中的具体技巧,内存管理的高效主要体现在内存的申请和释放效率。每次投递异步 I/O 时都必须创建新的句柄数据,数据空间的频繁创建和释放是相当占用系统资源的。这里可以使用内存池的方式来解决,基于 SLAB 算法实现内存池是一个好的思路,可以事先创建这些数据块的队列结构,并将其统一放入一个空闲队列,在投递 I/O 操作时,首先查看队列中是否有可用的空间:如有,则将第一个空间取出使用;否则真正向系统申请内存空间。在释放空间时,首先判断队列是否已满,如果未满,就将空间清空后从尾部存入队列;否则才真正释放这段空间。

同样在使用多个重叠 I/O 函数 AcceptEx 来守候大量短时连接请求时,可以事先创建一个 Socket 池,这就省去了连接时临时创建每个 Socket 的系统开销。

服务程序也可能对非分页池极限产生冲击。使用 IOCP 时会分配大量的套接字等内核对象,它们驻留在非分页池中而不会被交换到页面文件中。为了防止程序消耗到非分页池的极限,就要尽可能重用 Socket 等对象。比如使用 TransmitFile 和 TransmitPackets 函数时可以通过指定 TF_REUSE_SOCKET 和 TF_DISCONNECT 标志来重用套接字句柄,每当 API 完成数据的传输工作后,就会在传输层级别断开连接,这个套接字又可以重新提供给 AcceptEx( )使用。通过设置合理数量的 Socket 池和 Socket 重用,尽可能减少内存的动态申请和释放,提升了系统的性能。

2)资源释放与访问紊乱。

IOCP 资源释放是 IOCP 编程中的一大难点。如前面提及的在执行 WSAsend 函数出现非 Pending 的返回错误时,就要对此错误进行处理,包括释放客户端数据结构体和发送缓冲区数据,并关闭当前操作所使用的 Socket。但问题的焦点在于投递的 I/O 请求完成后都是放在一个 completion packet 队列里面等待 GetQuenedCompletionStatus 函数取出的,如果关闭 Socket 时把和 Socket 相关的资源都释放了,当客户端之前所执行的一些 I/O 调用返回,此时又试图去处理返回的 Completion Packet,一个访问紊乱(Access Violation)就发生了。

文献[10]给出了释放操作线性化的方法,但设计机制复杂,实现困难。可行的方法是对数据缓冲区使用引用计数机制。为了避免访问紊乱的发生,为客户端结构体增加一个阻塞 I/O 调用的计数,只有当计数为零,即不存在阻塞 I/O 调用时,才删除该结构体。

1.3　数据包重排序与字节块包处理

1.3.1　数据包重排序

IOCP 是个严格的先进先出(First In First Out, FIFO)系统,先提交的异步 I/O 请求先完成。当系统只有一个工作线程并且只在一个 Socket 投递一个异步操作时,该机制完美无缺;但是为了完全发挥 IOCP 的性能,可能会有多个工作线程在同一连接上同时投递多个异步操作,线程切换的随机性会导致缓冲区中的数据有可能顺序混乱。已有的文献中只强调了数据接收时数据包的排序处理[11-12],忽略了发送时的顺序问题。

全面的解决办法分为两个方面:1)针对发送端,调用 WSASend 的次序就是缓冲区被填充的次序,在多个工作线程的情况下不使用 PostQueuedCompletionStatus 向 IOCP 投递"写

数据”请求，因为线程切换的随机性会致使 WSASend 不一定按 Post 方法投递的顺序被调用，而导致对方接收到的数据可能乱序。同时避免从不同的线程中同时调用同一个 Socket 上的 WSASend 函数，避免缓冲区的数据处于不可预知的次序。2）使用顺序标签。在每个 Socket 上使用两个互斥变量分别来标识当前数据包读、写的顺序号，同时在单 I/O 数据块中设置重叠 I/O 操作的序号。在读取数据时，如果当前读取数据块的顺序号与I/O操作号相等，说明请求的数据即是现在要读取的数据；否则先缓存这个数据，等到它之前的数据包被处理后才能处理它；发送也采用相同的策略。这样就巧妙地解决了数据包的错序问题。

1.3.2 TCP 层引起的字节块中包的处理

TCP 协议会根据网络状况和 neagle 算法将一个逻辑包单独或分若干次发送。这样，接收到的一个字节块流可能包含一个或多个包，或者包的一部分，由此便带来了字节流数据块和部分包的拼装问题，如图 1 所示。

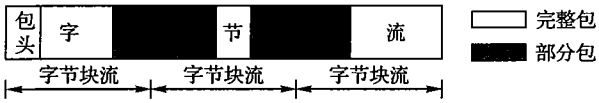


图 1 字节流块与数据包示意图

在图 1 中要想成功解读一个完整的应用层逻辑数据包，必须把若干半截的包按照包头的大小定义拼接到一起，拼包操作必然会涉及内存的拷贝操作。

为避免频繁地调用内存拷贝函数，可以采用“环形缓冲区”的方法。接收缓冲区使用一个 Byte 类型的数组来实现“环形队列”缓存区，使用 rear 和 front 两个指针分别来指示拷入和拷出数据的位置，通过指针的移动来实现数据的存放与读取。这样一次从缓冲区里取走所有完整逻辑数据包后，不用将缓冲区剩余的数据重新复制到首部以等待后续数据的拼装，而是根据记录来移动队列 rear 和 front 指针直接进行下一次的拼包操作，是一种优化方案。

2 IOCP 通信层设计与实现

2.1 总体设计

为了使程序具有通用性和可扩展性，通信层采用三层结构设计，分为 CIOCPServer、CProServer、TestServer 三层（见图 2）。针对第 1 章中讨论的 3 个 IOCP 关键问题和解决思路，对 CIOCPServer 和 CProServer 两层进行了优化和改进设计。

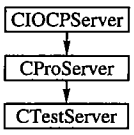


图 2 网络程序通信层三层结构

CIOCPServer 类对 IOCP 对象进行了封装，实现“完成端口服务器”基本通信功能：开启关闭服务器、管理客户端连接列表、管理未决的接受请求列表、顺序投递 I/O 异步操作、数据包的顺序读取等。同时通过多态机制向它的派生类提供客户端连接处理、数据收发完成处理、I/O 错误处理等接口。

CProServer 类继承 CIOCPServer，实现数据在网络传输与业务逻辑层的交互。数据包的结构定义、封装、解析，文本消息传递与文件传输均在此类中实现。如图 3 所示，主要包括接收、发送数据队列和相应的接收、发送线程，以及用户接口函数。CTestServer 是测试类。

2.2 网络通信层的实现

通信层的总体实现过程如图 3 所示。

1）使用 CreateIOCompletePort 函数创建 IOCP 对象，并使用 WSASocket 函数创建监听套接字，并将监听套接字关联到 IOCP 对象。

2）启动监听线程，负责投递重叠的 AcceptEx 接收操作。投递的 AccpetEX I/O 记录在链表里并调用 WSAWaitForMultipleEvents 函数等待网络事件发生，如果规定时间内等待超时，则使用 getsocketopt 函数遍历链表中每个未决的 AcceptEx I/O 操作，检查建立连接时间，超时则关闭这个连接，以避免恶意链接。

3）启动多个工作线程，负责读取并处理 IOCP 上的重叠 I/O 完成事件。如果是读操作完成，则按单句柄数据中的 m_nReadSequence 变量顺序读取，并通过接口函数 NotifyReadCompleted 通知用户，数据经处理后进入接收数据列表中。

4）启动多个数据处理线程，负责从接收数据队列中取出网络协议数据，并调用虚函数接口 ProcessMessage 来处理协议数据，放入发送数据列表中。对接收和发送列表的访问要互斥进行。

5）启动多个发送线程，负责从发送数据队列中取出数据并按照单句柄中的发送顺序号 m_nSendSequence 投递到 IOCP 上。

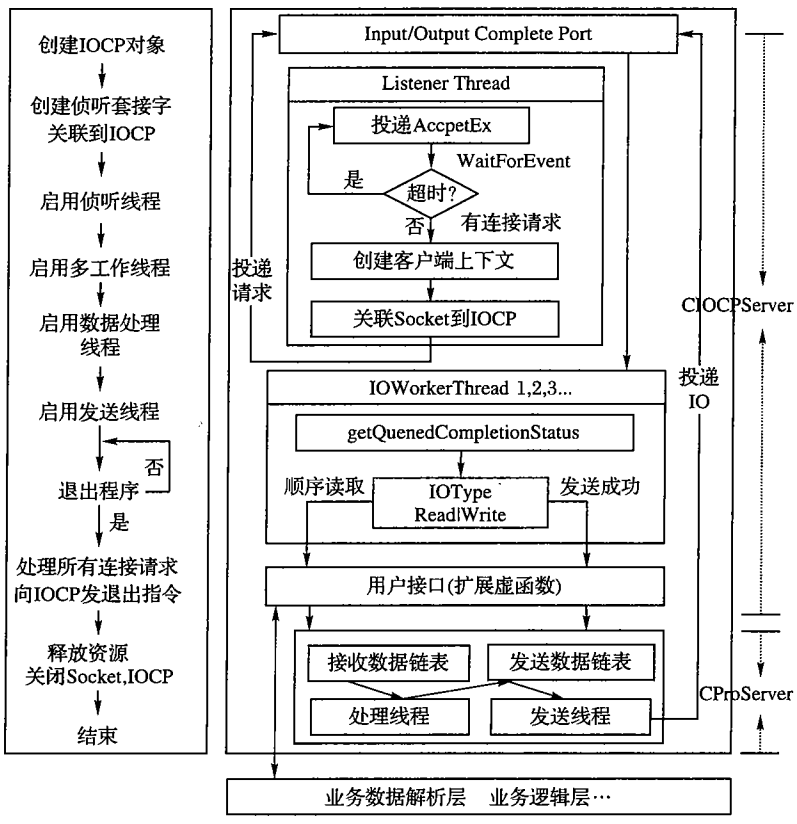


图 3 IOCP 通信层实现流程

6)重叠 I/O 缓冲区从内存池中申请和释放。使用指针 m_pFreeBufferList 指向缓冲区结构链表,申请时如果该指针值不为 NULL,则将第一个空间取出使用;释放时如缓冲链表未满则将空间清空后存入缓冲链表。

7)在退出时,通过单 I/O 数据中的计数参数 m_nNumberOfPendingIO 来监测连接上是否还有未决的 I/O,仅在所有未决 I/O 完成后才关闭 Socket 并释放资源。

3　性能测试与分析

3.1　测试说明和测试环境

本文主要从客户端并发量、响应情况和资源消耗程度 3 个方面对网络服务程序性能进行测试和分析。相对应地从最大支持的 Socket 连接个数、客户端饥饿数(在规定时间内得不到服务器响应的客户端个数)、单位时间内线程上下文切换次数、CPU 与内存消耗等 5 个指标进行度量。

运行通信程序的服务器配置为双 CPU,2.83 GHz,内存 2 GB;6 台客户机配置同为:AMD 64 CPU 2.1 GHz, 1 GB 内存的台式机,可以发起多达 6000 个 Socket 连接来模拟实际客户端的连接情况;网络环境为 100 Mbps 以太局域网。在测试程序中加入客户端饥饿判断和统计代码,同时实现了线程池模型的通信程序,用于对比测试。

3.2　测试结果分析

实验结果表明,IOCP 模型的服务程序基本上可以同时响应 3000 个以上的客户端。从表 1 可看出:由于 IOCP 模型采用先进先出的消息队列来处理服务,因此没有出现客户端得不到服务的情况;相反,线程模型随着客户端连接请求数目的增加,得不到响应的客户端也相应增多。从表 2 可看出:IOCP 只使用了固定数量的线程,线程的上下文切换次数在客户端数量不断增加时保持稳定;相反,后者分配的线程数随着客户端数量的增加而增加,相应带来了频繁的线程上下文切换,消耗大量的 CPU 和内存资源。

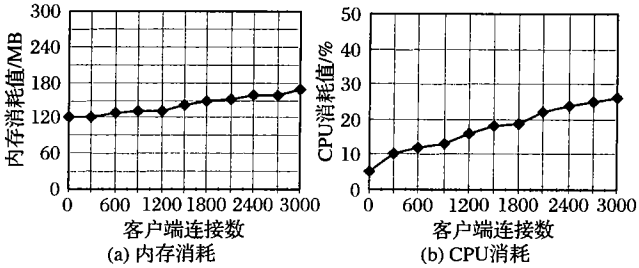
表 1　客户端饥饿数对比

模拟客户端数量	线程通信模型	IOCP 通信模型
50	0	0
100	7	0
150	11	0
200	22	0
250	31	0
300	42	0
500	51	0

表 2　线程上下文每秒切换次数对比

模拟客户端数量	线程通信模型	IOCP 通信模型
50	1080	781
100	2189	782
150	4960	789
200	5821	792
250	3710	794
500	3520	801
1000		806

图 4 为 IOCP 通信模型在客户端连接数量不断增加的情况下 CPU 和内存消耗情况,尽管程序一启动就占用了约 120 MB的内存,但随着客户端连接数量的增多,CPU 和内存消耗没有急剧增长,而是稳定地呈线性缓慢增长,表现出了良好的伸缩性能。



(a) 内存消耗　(b) CPU消耗

图 4　服务程序资源消耗测试结果

基于上述网络通信层实现了全国大学英语四六级网考模拟考试系统,在此基础上实现了客户端连接管理、下发题库、定时保存答卷、掉线自动重连、应急换机续考、试卷回收等业务功能。在上述服务器配置和网络环境中运行,约 1000 考生参加考试时,系统响应及时,运行稳定,性能表现良好。

4　结语

IOCP 提供了一种利用有限的资源公平地处理多客户端 I/O 请求问题的解决办法。本文通过使用"IOCP 参数扩展、内存池、对象池、环形缓冲"等改进的方法有效解决了 IOCP 开发中"信息识别与提取、资源管理、消息乱序处理"三个关键问题。实际应用表明,使用 ICOP 创建扩展性和高健壮的能够为大量客户服务的网络通信服务器是可行的。

参考文献:

[1] DESHPANDE J A. Windows sockets 2.0: Write scalable winsock apps using completion ports[EB/OL]. [2011-08-08]. http://msdn.microsoft.com/en-us/magazine/cc302334.aspx.

[2] RICHTER J, NASARRE C. Windows 核心编程[M]. 5 版. 葛子敖, 周靖, 廖敏, 译. 北京: 清华大学出版社, 2008.

[3] Microsoft. I/O completion ports [EB/OL]. [2011-09-10]. http://msdn.microsoft.com/en-us/library/windows/desktop/aa365198(v=vs.85).aspx.

[4] 叶锋, 陈和平, 肖金生. 基于 IOCP 的多串口与网络通信的实现[J]. 武汉理工大学学报, 2008, 30(2): 197 - 200.

[5] 金牧, 李文耀. IOCP 机制在 P2P 模式网络通信中的应用[J]. 微计算机信息, 2007, 23(8): 126 - 128.

[6] 蒋姝霞. 基于 IOCP 和规则分析的监控系统服务器实现机制研究[D]. 上海: 上海交通大学, 2010.

[7] ZHAO GEQI, WEI SHI, ZHAO HUIWU. Software architecture design on large-scale network traffic signal controllers system[C]// Proceedings of the 11th International IEEE Conference on Intelligent Transportation Systems. Piscataway, NJ: IEEE Press, 2008: 31 - 36.

[8] HOLGATE L. Handling multiple pending socket read and write operations[EB/OL]. [2011-08-10]. http://www.codeproject.com/KB/IP/reusablesocketserver4.aspx.

[9] 陈怀松, 陈家琪. IOCP 写服务程序时的关键问题研究[J]. 计算机工程与设计, 2010, 31(17): 3793 - 3796.

[10] 王文武, 赵卫东, 王志成, 等. 高性能服务器底层网络通信模块的设计方法[J]. 计算机工程, 2009, 35(3): 103 - 105.

[11] 王瑞彪, 李凤岐, 施玉勋, 等. 基于 IOCP 机制的网络游戏服务器通信层的实现[J]. 计算机工程与应用, 2009, 45(7): 75 - 78.

[12] 马金鑫, 袁丁. 基于 IOCP 的高并发通信服务器的设计与实现[J]. 通信技术, 2009, 45(7): 248 - 250.