计算机科学2002Vol. 29№. 11

用完成端口实现可扩展的服务器应用

Implement the Scalable Server Application with Completion Ports

吴 星 黄爱萍

(浙江大学信息与电子工程学系 杭州310027)

Abstract With the increasing variety of telecommunication business and needs for more and more client connections, the application servers are faced with the challenge of the overload of tremendous requests. On the Windows platform, components programming provides distribution structure at the business level. And at the user level, I/O completion port is a best way for the scalability. In this paper, we describe the way to use IOCP.

Keywords I/O completion port, Overlapped I/O, Scalability

基于 Socket 的网络服务应用已经使用得相当普遍,诸如创建 Socket、侦听/接受连接以及收发数据等方面也有很多经验文档和范例可供参考。然而实现可承受大数据量和连接数的网络服务应用程序还是一件比较困难的事情。最大的问题在于网络服务程序如何解决从单个客户连接处理到多个客户连接处理的可扩展性。

在 Windows NT 和 Windows 2000平台上,Overlapped I/O 运用"完成端口"(completion ports)来真正实现所谓可扩展的网络应用。"I/O 完成端口"(I/O completion ports)和 Windows Sockets 2.0可实现用于处理大量连接的应用。

本文将对基于完成端口的可扩展服务程序的实现进行讨论。

1 服务器应用对客户连接的管理

实现可扩展服务程序的关键在于服务器应用对客户连接的管理。

1.1 客户连接方式

客户程序和服务器应用程序之间通过 Socket 直接进行通信,服务器应用程序对客户连接的管理有5种方式:

1)单线程、单客户方式。这是最简单的方式,服务器程序在一个线程中处理单一客户的服务请求。

2)单线程、多客户方式。比前一种稍复杂,服务器程序在一个线程中处理多个客户连接的通信,因为每个 I/O 操作都必须通过 select 函数进行,性能较低。

3)每客户一个线程方式。这是最常用的方式,服务器程序有一个监听线程,并为每个客户连接创建一个通信线程。这种方式在连接数多的情况下因为线程切换会导致性能急剧下降。

4)同步 I/O 的工作线程方式。可以支持大量的连接数。服务器程序创建一个线程监听和监视客户连接,当某一连接需要服务时,将任务分发给工作线程完成。

5)异步 I/O 的工作线程方式。效率最高也是最复杂的方式,采用 IOCP(I/O Completion Port)作同步机制,在 Overlapped 模式下,服务器程序的每个工作线程都可以异步地处理多个客户的请求,这使得这种模式可以支持非常大量的客户连接数,而同时在连接数较小的情况下同样具有很高的效率。

1.2 优劣分析

对于前两种连接方式,从处理流程看属于串行模型。串行模型的问题在于它不能同时处理好多个请求。如果两个客户同时做出请求,一次只能处理一个请求,第二个请求必须等待第一个请求被处理完。显然,串行模型只适用于最简单的服务程序,在这里只有很少的客户请求,请求能被很快地处理。

第三种连接方式属于并发模型。由于串行模型的局限性,并发模型就使用得比较广泛。在并发模型中,对每个客户请求都创建了一个线程。其优点在于等待请求的线程只需做很少的工作。大多数时间中,该线程在休眠。当来了一个客户请求后,线程醒来,创建一个新线程来处理请求,而后等待下一个客户请求。这意味着客户请求能被快速地处理。而且,由于每个客户请求有其自己的线程,服务程序的伸缩性很好,因此在使用并发模型时,通过升级硬件(添加另一个 CPU),服务应用程序的性能可以得到提高。

当使用并发模型的服务应用程序实现在操作系统上时(此处的操作系统指 Windows NT 和 Windows 2000,以下同),这些应用程序的性能并没有预料的那么高。在处理很多同时的客户请求的情况下,就意味着会有很多线程并发地运行在系统中,而且所有这些线程都是可运行的。因此操作系统内核花费了太多的时间来转换运行线程的上下文,线程反而就无法得到足够的 CPU 时间来做它们的工作。

最后的两种连接方式都使用工作线程处理客户请求,具有较好的可扩展性。在 Win32平台的交叠(overlapped) I/O 机制中,允许一个应用程序先开始一个操作,异步接收其完成的通知,这一点对于那些耗时很长的操作非常有用。线程开始一个交叠 I/O 的操作后,可先去处理其它操作,然后再接收交叠操作的完成请求。显然,异步 I/O 的工作线程方式比同步 I/O 的工作线程方式的效率更高。然而前者在程序实现上较为复杂,幸运的是,Windows 平台提供了 I/O 完成端口这样一个内核对象,使得实现异步 I/O 的工作线程方式得以简化。

本课题中将采用第5种连接方式,以提供最大的可扩展性。

2 I/O 完成端口

在 Windows NT 和 Windows 2000平台上,唯一真正实现

吴 星 硕士生,主要研究方向为通信网管系统的设计。黄爱萍 教授,主要研究方向为移动通信、宽带无线接入、信号处理。

可扩展性的就是在 overlapped I/O 机制中运用完成端口处理通知消息。完成端口机制使得操作系统的内部工作交互达到最优化。

I/O 完成端口实际上是一个队列，操作系统将交叠 I/O 请求完成的通知放在这个队列中。一旦操作完成，完成通知就会唤醒工作线程处理结果。一个 socket 创建后就要有一个完成端口与其相关联。

应用程序一般都会创建一组工作线程，它们的数量取决于应用本身的需求。理想的数量为每个处理器一个线程，但是那样就意味着没有线程去处理象同步读写或事件等待这样的阻塞操作。每个线程都分配到一定的 CPU 时间。在这个时间中，线程可进行处理直到其他线程来获取此时间片。当一个线程处理阻塞操作时，操作系统就会将它的空闲时间片分配给其它线程。

并行模型的另一个低效之处是为每一个客户请求创建了一个新线程。创建线程比起创建进程来开销要小，但也远不是没有开销。如果当应用程序初始化时创建了一个线程池，而这些线程在应用程序执行期间是空闲的，应用程序的性能就能进一步提高。I/O 完成端口就使用线程池。

3 执行步骤

用完成端口实现与客户的 socket 通信分为三个步骤，第一步是初始化，创建完成端口、侦听线程和工作线程，第二步是侦听线程接收客户连接，创建 socket 并将其与完成端口相关联。第三步是工作线程接收到 I/O 完成信息后，进行相应的业务处理。

3.1 初始化及退出处理

主线程的流程图如图1所示。

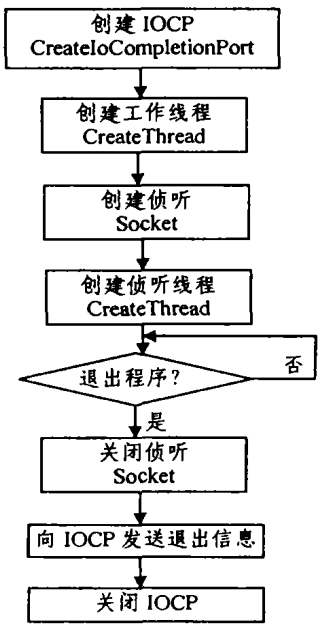

图1 主线程流程图

首先创建一个新的完成端口。这里用到的关键函数为 CreateIoCompletionPort。

```
HANDLE CreateIoCompletionPort (
HANDLE FileHandle, // handle to file
HANDLE ExistingCompletionPort, // handle to I/O completion port
ULONG_PTR CompletionKey, // completion key
DWORD NumberOfConcurrentThreads // number of threads to execute concurrently
);
```

其中 *FileHandle* 是与完成端口相关联的 Socket 句柄。*ExistingCompletionPort* 是已经存在的一个端口句柄，*CompletionKey* 一般用来存放与 *FileHandle* 相应的上下文信息。*NumberOfConcurrentThreads* 为允许对该完成端口进行操作的并发线程数。

这个函数有两种用法：当创建一个新的完成端口时，*FileHandle* 可为 NULL，返回值就是新端口的句柄。而当需要将一个已存在的 Socket 句柄与端口相关联时，*ExistingCompletionPort* 表示端口句柄，*FileHandle* 和 *CompletionKey* 为 Socket 句柄及该 Socket 对应的上下文信息。

然后创建工作线程和侦听线程，参照 Jeffery Richard 的建议，工作线程的个数为应用程序所运行平台 CPU 个数的2倍。

服务程序退出时，应关闭侦听的 Socket，使侦听线程退出。向完成端口发送退出的信息包，通知工作线程退出。

3.2 侦听连接

侦听线程的流程图如图2所示。

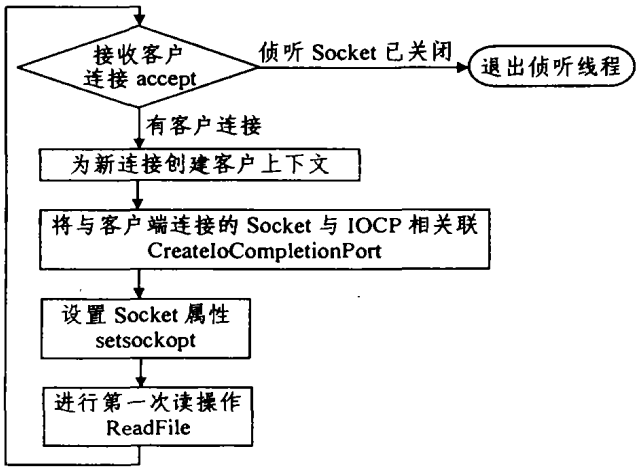

图2 侦听线程流程图

侦听线程中，调用了 accept 函数来接收客户连接。当没有客户连接时，该函数进入阻塞状态，一旦有客户连接发生，返回一个新的 socket。分配一个客户上下文信息的结构，把这个新的 socket 和上下文信息的结构通过 CreateIoCompletionPort 与完成端口关联起来。由于现在已经存在着一个 Socket 句柄，因此这次调用 CreateIoCompletionPort 函数时参照3.1节中的第二种用法。然后用 ReadFile 触发新 Socket 上的第一次读操作。当接收到客户请求时，系统将读完成信息包发送到完成端口，由工作线程进行相关的业务处理。

典型的上下文信息结构中一般包括该 Socket 的读写状态，读写缓冲等信息，具体数据结构见下：

```
class ClientContext
{
public:
    OVERLAPPEDPLUS m_OverlappedRead;
        //读操作使用的 OVERLAPPEDPLUS 结构
    OVERLAPPEDPLUS m_OverlappedWrite;
        //写操作使用的 OVERLAPPEDPLUS 结构
    SOCKET m_Socket; //与客户连接的 Socket
    // Store buffers
    CBuffer m_ReadBuffer; //读缓冲
    CBuffer m_WriteBuffer; //写缓冲
    CBuffer m_SendingBuffer; //当前正在写的缓冲
    HANDLE m_hWriteComplete; //写完成事件句柄
```

（下转第164页）

供应集合组成了它的交易空间。最后，客户机通过执行查询在交易空间中查找对象。

以上方法复杂度不同，在应用时主要考虑的是需要公布对象的数量，客户机查找对象的标准，用户需要的服务质量等等因素。对于很简单的静态环境，使用对象引用字符串是合适的，因为它不需要任何的附加服务，客户机只需要使用对象引用字符串就可以连接到合适的服务器。对于复杂环境下，命名服务和交易服务是优先选择，它们有细小的差别。命名服务有固定的多维层次结构，需要绑定的每个对象只有它的名字和在层次中的位置两部分关联信息，能够实现高效的查询。但是命名服务只能返回所需的单独一个对象，而且在命名层次发生变化时会对客户机应用程序影响较大。相比较而言，交易服务是二维的，每个服务类型可以有任意数量的属性，服务类型的增加不会影响已经存在的供应。在客户机使用变化的标准来查找对象的时候，交易服务就是更好的选择。

结论 本文主要在对远程过程调用和远程对象调用做了简单对比的基础上，对在 DCOM 和 CORBA 中获取远程对象的方法进行分析和研究。CORBA 和 DCOM 中接口的概念是一致的，都是把客户与组件分离，把接口与实现分离，组件与其客户之间通过细粒度的接口进行交互。CORBA 中各种语言的映射机制实现其语言的无关性，DCOM 中的语言无关性建立在面向对象程序设计中的纯虚函数概念的基础上，凡是符合该规范的语言都可以实现 COM 组件。

它们都提出对象工厂的概念，通过对象工厂来找到或者建立所需的远程对象。CORBA 中的对象工厂和 COM 中的类工厂相似，用来创建一个我们实际使用的对象。COM 中的类工厂只能找到特定的对象引用，而 CORBA 中的对象工厂还可以返回其它的对象，主要用来返回已存在的大量的对象引用。

它们都有一定的命名解析机制，在 CORBA 中使用较多的是命名服务，通过层次化的命名空间来解决命名冲突和实现对象的有效管理。在 DCOM 中相对应的是解析器的概念，它负责管理对象输出标识符表格。在传输远程对象所需要的参数的机制中，它们都使用打包(marshal)机制，只是具体的打包方法不同。

可以预见，本文中所论述的方法是未来分布式计算和分布式系统中获取远程对象的必然选择。

参 考 文 献

1 Object Management Group. Common Object Request Broker: Architecture and Specification, 2001

2 Object Management Group. CORBA services: Common Object Services Specification, 1998

3 Vinoski. New Features for CORBA 3.0. Communications of ACM, 1998, 41(10)

4 Schmidt D C. The Design and Performance of Real-Time Object Request Brokers. Computer Communication, 1998, 21(4): 294～324

5 潘爱民. COM 原理与应用. 北京：清华大学出版社，1999

6 Slama D, garbis J, russell P. enterprise CORBA. Prentice Hall, 1999

(上接第145页)

```
    // Input Elements for Winsock
    WSABUF m_wsaInBuffer; // WSARecv 使用的接收缓冲
    BYTE m_byInBuffer[8192];
    // Output elements for Winsock
    WSABUFm_wsaOutBuffer; // WSASend 使用的发送缓冲
    CRITICAL_SECTION m_csWriteBuffer; // 保护写缓冲的临界量
};
```

3.3 工作线程

工作线程的流程图如图3所示。

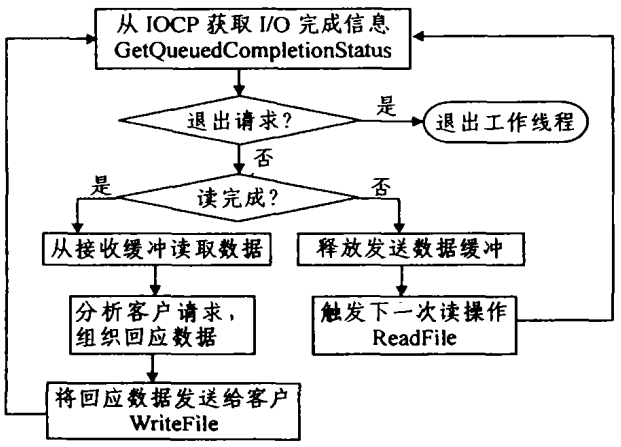

图3 工作线程流程图

一旦完成端口已创建并与 sockets 进行关联，一个或多个工作线程就可以进行完成通知的处理。工作线程调用 GetQueuedCompletionStatus 函数从 I/O 完成端口队列取得完成通知，从其第三个参数 lpCompletionKey 指向的客户上下文信息中获取与 socket 相关的上下文信息，判断其完成通知状态，进行相应的业务处理。如果发现完成信息包是退出指示，则结束线程。

```
BOOL GetQueuedCompletionStatus(
  HANDLE CompletionPort // handle to completion port
  LPDWORD lpNumberOfBytes, // bytes transferred
  PULONG_PTR lpCompletionKey, // file completion key
  LPOVERLAPPED * lpOverlapped, // buffer
  DWORD dwMilliseconds // optional timeout value
);
```

结论 基于 I/O 完成端口技术，服务器应用可以得到很好的可扩展性。本文比较了服务器应用对客户连接各种处理方式的优劣，描述了 IOCP 的工作原理，并在此基础上较为全面地分析和介绍了实现 IOCP 的各个步骤，并且详细描述了其中关键函数的用法。

微软在 Windows NT 和 Windows 2000上提供了 IOCP，并且在 IIS 中采用这种技术处理客户请求，得到了很好的效果。可以预见，将会有越来越多的服务器应用采用 IOCP 技术。

参 考 文 献

1 Reilly D J. 基于服务器的应用程序内幕. Microsoft Press，北京大学出版社，2000.9

2 Richard J. Windows NT 核心技术. 清华大学出版社