基于 IOCP 的服务器端应用程序①

Sever-Side Application Based on IOCP

杜 翔 雷跃明 (重庆大学 软件学院 重庆 400044)

摘 要: 本文介绍了 IOCP（I/O Completion Port 输入/输出完成端口）的基本原理，IOCP 是一种能够合理利用与管理多线程的机制，可以帮助处理大量客户端请求的网络服务问题，是 Windows 系统平台上用于开发高性能的服务器端应用程序的最好的 I/O 模型。本文最后结合实践给出了一个基于 IOCP 开发服务器端应用程序的设计方案和其部分实现代码。

关键词: 完成端口 异步 I/O 线程池 线程同步 服务器端应用程序

1 引言

要想编写一个高性能的服务器应用程序，必须实现一个高效的线程模型。让太少或者太多的服务器线程来处理客户的请求，都可能会导致性能问题。构造一个服务器端应用程序通常你可能会采用以下两种模型之一：

①串行化模型 - 一个线程等待客户端连接，当客户请求到达时，该线程醒来开始处理客户请求。

②并行化模型 - 一个线程等待客户端连接，当客户请求到达时，该线程创建新的新的工作线程去处理客户请求。

串行化模型的问题是它无法处理同时来的多个请求。如果两个客户同时发出请求，那么只有第一个能得到处理，第二个则必须等待直到第一个处理完毕。一个基于串行化模型设计的服务器无法获得多处理器系统的优势。

在并行化模型里，每一个线程都对应一个客户请求。这样做的优点是每个新到达的客户请求都能很快得到处理，并且可以很方便的利用多处理器系统的优势。但工作线程的数目会随着客户请求的数量而不断增多，从而导致大量额外的环境切换（context switching），这是由于调度器必须将处理器时间在多个活动线程之间划分而引起的。

Windows NT 3.5 中引入了 IOCP 管理内核对象，IOCP(I/O Completion Port 输入/输出完成端口)是一种能够合理利用与管理多线程的机制，IOCP 的理论基础是：并行运行的线程数目必须有一个上限。这样可以有效避免当线程数目过多而进行环境切换时所造成的 CPU 浪费。IOCP 模型是到现在为止在性能和可伸缩性方面表现最好的 I/O 模型。

2 IOCP 介绍

IOCP 模式要求创建一个 win32 IOCP 对象来对异步 I/O 请求进行管理，并通过创建一定数量的工作线程(work thread)，来为已经完成的异步 I/O 请求提供服务。其次，可以把 IOCP 看成系统维护的一个队列，操作系统把异步 I/O 操作完成的事件通知放入该队列，由于是“操作完成”的事件通知，故取名为“完成端口”。一个 IOCP 对象被创建以后，可以和多个套接字句柄进行关联，并在关联后的套接字上进行异步 I/O 操作。当 I/O 操作完成后，一个 I/O 完成的事件通知就会被排在此端口的完成队列上，此时，某个工作线程将会被唤醒来为 IOCP 服务，执行特定的处理工作。IOCP 的操作示意图如下：

3 使用IOCP

3.1 创建 IOCP

使用 IOCP 模型，首先要通过调用 Windows API 函数 CreateIoCompletionPort 创建 IOCP 对象，然后我们可以使用此对象来为任意数量的套接字句柄管

① 收稿时间:2008-08-06

理其对应的 I/O 请求，函数的原型如下：

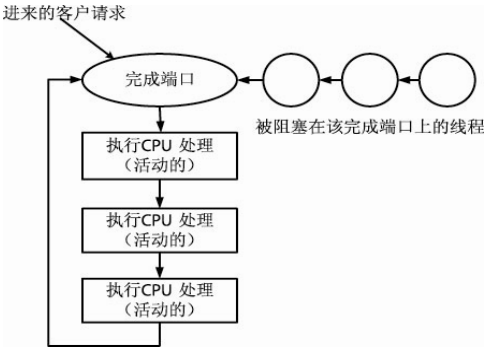

图 1　IOCP 的操作

```
HANDLE CreateIoCompletionPort(
  // 文件句柄
  HANDLE   FileHandle,
  // IOCP 句柄
  HANDLE   ExistingCompletionPort,
  // 完成键值
  ULONG_PTR   CompletionKey,
  // 最大并发线程数
  DWORD   ConcurrentThreadsNum
);
```

该函数执行两项任务：首先用它创建一个 IOCP 对象，然后再次调用将一个设备句柄和该 IOCP 对象绑定起来。为了方便在编程中的使用，可以将该函数的两个功能重新封装成两个函数，函数的实现如下：

```
// 1. 创建完成端口对象
HANDLE CreateNewCompletionPort(
  DWORD dwNumberOfConcurrentThreads)
{
  return CreateIoCompletionPort(
            INVALID_HANDLE_VALUE, NULL, 0,
            dwNumberOfConcurrentThreads));
}
// 2. 将设备和完成端口对象绑定
BOOL AssociateDeviceWithCompletionPort(
  HANDLE hCompPort, HANDLE hDevice,
  DWORD dwCompKey)
{
  HANDLE h = CreateIoCompletionPort(
                hDevice,hCompPort,dwCompKey,
0);
    return (h = = hCompPort);
}
```

第一个函数是 CreateNewCompletionPort，用于创建 IOCP 对象。函数唯一需要指定的参数是 dwNumberOfConcurrentThreads，它定义了允许在完成端口上同时执行的线程的数量。在理想情况下，我们希望每个处理器仅允许一个线程来为完成端口提供服务，以避免线程进行上下文切换。将 0 作为 dwNumberOfConcurrentThreads 参数可以达到此目的；第二个函数是 AssociateDeviceWithComple--tionPort，用于将套接字（socket）和已创建的完成端口关联起来。dwCompKey 参数被称为完成键，此数据参数可以传递一个指向包含 socket 信息结构体的指针，在每次完成通知到达时，该信息结构体也会随通知一起被收到。

3.2 工作线程池

在将套接字与 IOCP 对象关联之前，需要先创建一个工作线程池，它们将在该完成端口等待处理投递到该端口上的 I/O 请求。在前面一节提到过，在创建 IOCP 对象时，需要指定一个并发性值。该值指示了在任何给定时候正在运行的、与该端口相关联的线程的最大数量。Windows 利用该值来控制一个应用程序可以有多少个活动的线程。如果与一个端口相关联的活动线程的数量等于此并发性值，那么，正在该完成端口上的等待的线程便不允许再允许了。而当一个活动线程在处理过程中被阻塞时，比如需要对磁盘上的文件进行读写数据的情况，Windows 会检测到这一行为，如果此时完成端口的队列中有包的话，则会唤醒另一个正在等待该完成端口的线程。综上所述，工作线程池由以下几种线程组成：

①正在执行的线程

②被阻塞的线程

③在完成端口上等待的线程

正因如此，应该创建比并发线程数更多的线程。Windows 的指导原则是，将并发性值设置成大约等于该系统中处理器的数目，而合理的工作线程的个数则应该 CPU 的个数的两倍再加 2。需要注意的是，线程并不是免费的，更多等待在完成端口上的线程并不会让系统速度更快一些。

3.3 线程同步机制

工作线程池创建后，每个线程就可以通过调用

GetQueuedCompletionStatus 函数与完成端口关联起来。凡是调用该函数的线程都将被放入该完成端口的等待线程队列中，由完成端口来维护管理。当端口上有完成包(Completion Packet)到达时，这些线程将按照后进先出（LIFO）的顺序被唤醒，并由 GetQueuedCompletionStatus 函数获得该完成包，开始处理 I/O 请求。GetQueuedCompletionStatus 函数的原型如下：

```
BOOL GetQueuedCompletionStatus (
  // IOCP 句柄
  HANDLE  CompletionPort,
  // 传输字节数
  LPDWORD  lpNumberOfBytes,
  // 完成键值（套接字信息结构体指针）
  PULONG_PTR  lpCompletionKey,
// 重叠结构体指针（I/O 操作信息结构体指针）
LPOVERLAPPED  *lpOverlapped,
// 超时时间
  DWORD  dwMilliseconds
);
```

其中最重要的两个参数，lpCompletionKey 参数和 lpOverlapped 参数都是一个指针，我们可以通过向这两个参数传入一个自定义信息结构体的地址值，以此来实现线程间数据的同步和共享，工作线程也可以由这两个参数来取得有 I/O 事件发生的套接字的相关信息。

lpCompletionKey 参数是在调用 AssociateDeviceWithCompletionPort 函数将设备句柄关联到完成端口时传入的，与设备是一一对应的关系，因此我们可以将与套接字相关的信息结构体的地址作为此参数传入函数。套接字信息结构体的定义如下：

```
// 1. 关联到完成端口的每个套接字的相关信息
typedef struct _PER_SOCKET_CONTEXT {
  // 套接字句柄
  SOCKET  Socket;
  // I/O 操作信息结构体的链表
  PPER_IO_CONTEXT  lpIOContext;
  // 套接字信息链表后向指针
  struct _PER_SOCKET_CONTEXT  *lpCtxtBack;
  // 套接字信息链表前向指针
    struct_PER_SOCKET_CONTEXT*lpCtxtForwa
rd;
}PER_SOCKET_CONTEXT,*PPER_SOCKET_CONTE
XT;
// 2. 包含所有套接字信息结构体的全局链表
PPER_SOCKET_CONTEXT g_pCtxtList = NULL;
```

lpOverlapped 参数在调用异步 I/O 函数 WSASend/WSARecv 时传入，与每次 I/O 操作是一一对应的，通常用来传递异步 I/O 请求所使用的内存缓存。I/O 操作信息结构体的定义如下：

```
// 1. I/O 操作类型
typedef enum _IO_OPERATION {
  ClientIoAccept,  // 新的连接
  ClientIoRead,    // 读操作
  ClientIoWrite    // 写操作
} IO_OPERATION, *PIO_OPERATION;
// 2. 套接字上每个 I/O 操作的相关信息
typedef struct _PER_IO_CONTEXT {
  // 重叠结构
  WSAOVERLAPPED  Overlapped;
  // 数据缓冲区
  CHAR  Buffer[4096];
  // 数据缓冲参数
  WSABUF  wsabuf;
  // 缓冲区中的数据总量
  int  nTotalBytes;
  // 已发送的数据量
  int  nSentBytes;
  // 当前的 I/O 操作类型
  IO_OPERATION  IOOperation;
  // I/O 操作信息链表前向指针
    struct_PER_IO_CONTEXT*lpIOContextForwa
rd;
}PER_IO_CONTEXT, *PPER_IO_CONTEXT;
```

需要注意的是，这里用到一个小技巧 – 重叠结构体 Overlapped 成员必须置于第一位，这样扩展结构体实例的地址才能作为重叠结构体的地址参数传给异步 I/O 函数。

3.4 关闭 IOCP

通过调用函数 PostQueuedCompletionStatus 服务器应用程序可以将自己定义的完成包排队到一个

完成端口中。因此，在程序结束时，可以先调用此函数(N 次，N 等于工作线程的数量)来提交一个特殊的完成包(这里我们向 dwCompletionKey 参数传递 0)，通知所有的工作线程立即退出，然后关闭所有的套接字并释放对应的信息结构体，最后关闭完成端口。PostQueuedCompletionStatus 函数的原型如下：

```
BOOL PostQueuedCompletionStatus(
  // 完成端口句柄
  HANDLE  CompletionPort,
  // 传输字节数
  DWORD  dwTransBytesNum,
  // 完成键值（套接字信息结构体指针）
  ULONG_PTR  dwCompletionKey,
// 重叠结构体指针（I/O 操作信息结构体指针）
LPOVERLAPPED  lpOverlapped
);
```

4　程序流程

采用 IOCP 模型为服务器端应用程序的网络接口部分提供服务可以获得最佳的系统性能。程序中有两种类型的线程 – 主线程和工作线程。主线程负责创建并监听套接字，创建工作线程，等待并接受到来的连接，并关联到 IOCP 等；而工作线程则负责等待并处理在 IOCP 对象上完成的 I/O 事件。程序的设计流程如下：

①调用函数 CreateNewCompletionPort 创建一个 I/O 完成端口；

②创建工作线程池。

③工作线程循环调用函数 GetQueuedCompL–-etionStatus 以获取 I/O 操作结果；

④主线程循环调用 accept 函数等待客户端请求连接；

⑤当有客户请求连接时，主线程的 accept 函数返回，建立新连接，并将套接字句柄用 As(六)sociateDeviceWithCompletionPort 函数关联到完成端口，然后发出一个异步的 WSASend/WSARecv 调用。因为是异步函数，WSASend/WSARecv 会直接返回，实际的发送或接受数据的操作由 Windows 系统完成；

⑥主线程继续下一次循环，阻塞在 accept 函数处等待新的客户端连接；

⑦Windows 系统完成 WSASend/WSARecv 操作，并将结果发至完成端口；

⑧工作线程的 GetQueuedCompletionStatus 函数立即返回，从完成端口取得 WSASend/WSARecv 操作的结果；

⑨工作线程对结果数据进行处理，接着继续调用 WSASend/WSARecv 函数，继续下一次循环并阻塞在函数 GetQueuedCompletionStatus 处；

上述处理流程如下图所示，其中虚线表示 Windows 系统进行的处理，不需要程序干预。

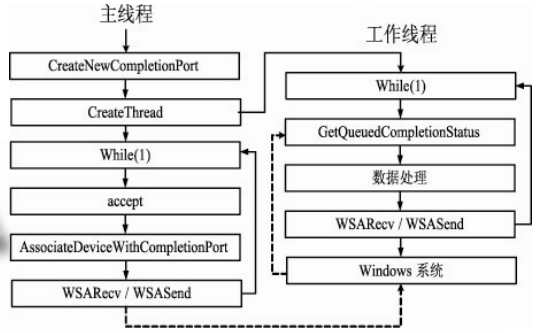

图 2　I/O 完成端口处理流程

5　总结

IOCP 模型，是迄今为止最为复杂的一种 I/O 模型。该模型适合在 Windows NT 或 Windows 2000 下开发高性能的服务器应用时使用。特别是在应用程序需要同时管理数百乃至数千个套接字，并且希望随系统 CPU 数量增加，应用程序的性能也能随之提高时，采用这种模型可以达到最佳系统性能。

参考文献

1 王艳平,张越.Windows 网络与通信程序设计.北京:人民邮电出版社, 2006.53－99.

2 Russinovich ME, Solomon DA. 潘爱民,译.深入解析 Windows 操作系统.第四版,北京:电子工业出版社, 2007:585－589.

3 Richter J, Clark JD.Programming Server-Side Application for Microsoft Windows 2000. Microsoft Press, 2000:10－35.

4 Douglas E, Comer D, Stevens L. 张卫,王能,译.TCP/IP 网络互联技术(卷 3):客户-服务器编程与应用(Windows 套接字版).北京:清华大学出版社,2004:17－30.

5 Beveridge J, Wiener R. 侯捷译.Win32 多线程程序设计.武汉:华中科技大学出版社, 2006.149－190.