◎研发、设计、测试◎

基于IOCP机制的网络游戏服务器通信层的实现

王瑞彪,李凤岐,施玉勋,张宪超
WANG Rui-biao,LI Feng-qi,SHI Yu-xun,ZHANG Xian-chao

大连理工大学 软件学院,辽宁 大连 116621
School of software,Dalian University of Technology,Dalian,Liaoning 116621,China
E-mail:wrtrtew_re@yahoo.com.cn

WANG Rui-biao,LI Feng-qi,SHI Yu-xun,et al.Realization of IOCP-based net layer of online games server.Computer Engineering and Applications,2009,45(7):75-78.

Abstract: This paper illustrates the development of online games server by using an efficient Input/Output method -IOCP mechanism provided by the Windows operating system.It gives the realization of the net layer of online games server,and supplies the game developer with virtual function interfaces.So,the game developer can concentrate on developing the game logic.Finally,the method is presented to apply these interfaces.

Key words: Input Output Completion Port(IOCP);online games;server;interface

摘 要:以网络游戏服务器的开发为背景,利用Windows操作系统提供的一种高效I/O机制-IOCP(Input Output Compeltion Port)机制,给出了实现网络游戏服务器端网络通信层的方法,并留虚函数接口给游戏开发者,使游戏开发者专注于游戏逻辑的开发,不再考虑网络通信的问题。最后,结合开发的一个象棋网络游戏服务器给出了使用这些接口的方法。

关键词:完成端口;网络游戏;服务器;接口
DOI:10.3778/j.issn.1002-8331.2009.07.024 **文章编号**:1002-8331(2009)07-0075-04 **文献标识码**:A **中图分类号**:TP391

1 引言

近年来,网络游戏在我国发展迅速,越来越多的开发商把目光投向了网络游戏开发。开发网络游戏最难的部分是如何设计一个高效的游戏服务器,使其尽最大可能为更多的客户提供服务。与此同时,使它具有最大的可扩展性和高稳定性,所谓的可扩展性,就是它可以满足少量用户访问时的需求,同时也可以满足大量客户并发访问时的需求。所谓的高稳定性,就是它可以长时间地为网络用户提供服务而不会发生错误。而这些性能,在很大程度上取决与网络通信层的性能。所以,开发一个高效的网络通信层就显的格外重要了。

作为一个研究开发人员,关心的是怎样开发,可以使开发出的服务器具有上述优势,然而由于商业上的原因,这方面的资料不是很多,即使有一些文献讲述了服务器的开发,但都是给一个整体的架构,只是说应该用IOCP机制去开发,至于怎么开发没有讲述。

所以作者根据自己多年在网络通信开发方面的经验,以网络游戏服务器开发为背景,给出利用Windows操作系统提供的IOCP机制,设计和开发网络游戏服务器网络通信层的方法,并留虚函数接口给游戏开发者,使游戏开发者专注于服务器游戏逻辑的开发,不再考虑网络通信的问题。最后,结合开发的一个象棋网络游戏服务器给出了使用这些接口的方法。

2 IOCP机制

2.1 IOCP及其工作原理

IOCP(Input Output Completion Port)是Windows操作系统提供的一种高效的I/O模型。它是应用程序使用少量工作线程来处理大量异步I/O请求的一种机制。

可以简单地把IOCP看成是系统维护的一个队列,操作系统把重叠I/O操作完成的事件通知放入该队列。然后通知线程来处理相应的消息。一个IOCP创建以后,可以和多个文件句柄进行关联。然后在这些文件句柄上进行重叠I/O操作。当一个I/O操作完成以后,一个重叠I/O完成的事件通知就会被放进此端口的完成队列上,此时,某个工作线程就会被唤醒,来执行特定的处理工作[1]。

2.2 IOCP机制的优势

与其他I/O模型相比,IOCP最大的优点在于其管理海量连接时的处理效率。它可以使用少量的线程来为大量的客户服务,充分利用系统资源。

使用其他的套接字I/O模型,如WSAAsyncSelect I/O模型,WSAEventSelect I/O模型,Overlapped(重叠)I/O模型,在处理海量连接时,系统需要启动大量的线程来为客户提供服务。但是如果同时有很多线程一起执行,系统的性能会马上下降很多,原因就是操作系统需要不断地进行上下文切换,耗费了大

作者简介:王瑞彪(1985-),男,工程师,主要研究方向:网络通信、图形图像处理;李凤岐(1974-),男,硕士,主要研究方向:网络通信;施玉勋(1960-),男,博士,教授,主要研究方向:远程教学,资讯系统安全;张宪超(1971-),男,博士,副教授,主要研究方向:智能搜索。

收稿日期:2008-01-02 **修回日期**:2008-03-31

量的资源和时间。

使用 IOCP 机制，可以避免使用大量线程，大大提高了系统的性能。IOCP 机制是目前在 Windows 平台下伸缩性能最好的一种 I/O 模型。

2.3 IOCP 使用方法[2]

(1)创建 IOCP 对象。可以使用 Windows 操作系统提供的 API 函数 CreateIoCompletionPort 来创建一个 IOCP 对象。

函数定义如下：

```
HANDLE CreateIoCompletionPort(HANDLE FileHandle,
    HANDLE ExistingCompletionPort,
    ULONG_PTR CompletionKey,
    DWORD NumberOfConcurrentThreads);
```

设定第 4 个参数，它定义了允许在 IOCP 上同时执行的线程数量。如果将第 4 个参数设置成 0，表示系统允许同时在 IOCP 上执行的线程数量与处理器数量一样多。

(2)创建 I/O 服务线程。

(3)关联套接字句柄到 IOCP。可以使用函数 CreateIoCompletionPort 将一个套接字句柄关联到 IOCP。

(4)处理完成的重叠 I/O 操作。当重叠的 I/O 操作完成时，I/O 系统会向 IOCP 对象发送一个完成通知封包，IOCP 以先进先出的方式为这些封包排队。在 I/O 服务线程中可以使用函数 GetQueuedCompletionStatus 来取得队列中的这些封包。

函数定义如下：

```
BOOL GetQueuedCompletionStatus(HANDLE CompletionPort,
    LPDWORD lpNumberOfBytes,
    PULONG_PTR lpCompletionKey,
    LPOVERLAPPED *lpOverlapped,
    DWORD dwMilliseconds);
```

通过调用 GetQueuedCompletionStatus 函数可以取得有事件发生的套接字的信息。通过 lpCompletionKey 参数可以得到与套接字句柄关联的数据，这个数据就是在关联套接字时传给函数 CreateIoCompletionPort 中的第三个参数 CompletionKey 的数据。通过 lpOverlapped 参数可以得到投递 I/O 请求时使用的重叠对象地址。

3 网络游戏服务器网络通信层的设计与实现

3.1 网络游戏服务器网络通信层的设计

3.1.1 句柄唯一数据(客户上下文)

所谓的句柄唯一数据就是与某一个套接字句柄密切相关的各参数的集合，可以把它设计成如下结构。

```
Struct CClientContext
{
  SOCKET m_hSocket;          //套接字句柄
  nCurrentReadSequence;      //当前要读的序列号
  nCurrentSendSequence;      //当前要发送的序列号

  ……        //其他相关参数
};
```

这个结构在关联套接字句柄到 IOCP 时要用到，传给 CreateIoCompletionPort 函数的第 3 个参数 dwCompletionKey 的数据，其实就是指向这个结构的一个指针，当使用 GetQueuedCompletionStatus 检查消息对列时，参数 lpCompletionKey 的返回值也是指向这个结构的指针。

3.1.2 I/O 唯一数据(缓冲区)

所谓的 I/O 唯一数据就是由投递重叠操作时要使用的缓冲区和重叠对象组成的集合。可以把它设计成如下结构。

```
Struct CClientBuffer
{
  WSAOVERLAPPED ol; //重叠结构
  Char *buff; //缓冲区,在这里要存放游戏消息(Game Message)
  Int nSequenceNumber; //操作的序列号

  ……    //其他相关参数
};
```

这个结构在进行投递重叠操作时要用到。当使用 GetQueuedCompletionStatus 检查消息对列时，由参数 lpOverlapped 的返回值可以得到这个结构，进而从缓存区 buff 中取得要操作的数据的内容。

CClientBuffer 结构中的参数 nSequenceNumber 和 CClientContext 结构中的参数 nCurrentReadSequence 和 nCurrentSendSequence 是为了解决数据包可能不按照正确顺序读取或发送而设计的。

由于有多个线程同时在 IOCP 上执行任务，即使投递在 IOCP 上的 I/O 操作总是按照被提交的顺序完成，但是线程的调度问题，可能会导致关联在 IOCP 上的操作不能按照正常的顺序完成。比如说，同时有两个线程在 IOCP 上执行任务，应该读取数据的顺序是“数据 1，数据 2，数据 3”，但可能由于线程调度问题，错误的以“数据 2，数据 3，数据 1”的顺序读取，这是不允许的。同样的，在 IOCP 上投递发送数据请求时，也可能会按照不正确的顺序投递，从而导致以不正确的顺序发送数据，这对要求按正确顺序发送的客户端来说是不允许的。

根据上面的设计，在实现的时候，如果线程读取数据，要按照 CClientContext 结构中的 nCurrentReadSequence 顺序读取，如果线程投递发送请求，则要按照 CClientContext 中的 nCurrentSendSequence 顺序投递。这样就可以巧妙解决这种错序问题。

3.1.3 总体结构

大部分的网络通信程序，都是按照分层的思想来设计开发[3]。这里也采用了这种思想，对于网络游戏，一般都按照下面的方法来分层，如图 1，分为游戏逻辑处理层、游戏协议解析层、网络通信层[4]。这里关注的是怎么开发网络通信层。网络通信层的架构如图 2。

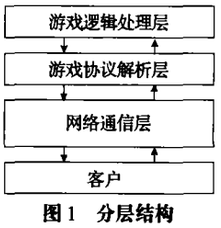

图 1 分层结构

对网络通信层中各个部分的功能作如下设计：

(1)监听线程负责投递异步的 AcceptEx I/O 操作。

(2)服务线程负责处理完成的重叠 I/O 操作，并将处理的结果通过虚函数接口通知用户处理。

(3)接收数据列表存放服务线程从 IOCP 上读取的数据，其实就是游戏协议数据，之所以要通过用户接口，是因为考虑

到用户可能要进行预处理。

(4)发送数据列表存放经数据处理线程处理过的游戏协议数据。

(5)数据处理线程调用虚函数接口中的游戏数据处理函数,不断的从接收数据链表中取出数据进行处理,并将处理的结果放进发送数据链表。

(6)发送线程不断的从发送数据列表中取出数据,将数据投递到IOCP上。

(7)对接收数据链表和发送数据链表的访问要互斥进行,以防止多个线程并发访问时发生错误。

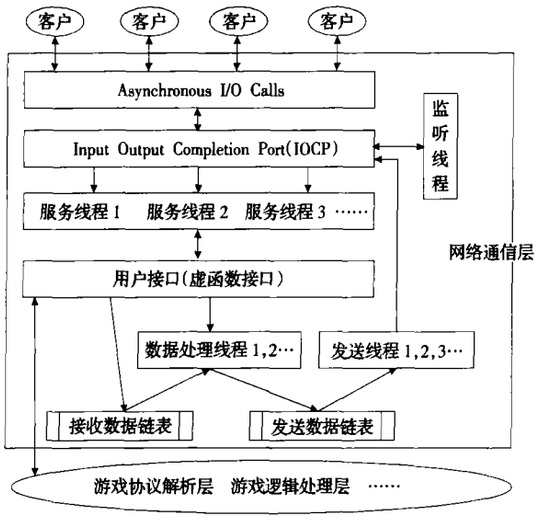

图2　网络游戏服务器网络通信层架构图

3.2　服务器网络通信层的实现

3.2.1　总体实现步骤

(1)使用CreateIoCompletionPort函数创建IOCP对象,并设定允许同时在IOCP上执行的线程数量为服务器的处理器数量。

(2)使用WSASocket函数创建监听套接字,并使用CreateIoCompletionPort函数将监听套接字关联到IOCP对象。

(3)启动监听线程,负责投递重叠的接收I/O(AcceptEx I/O)操作。

(4)启动多个服务线程,负责在IOCP上处理重叠I/O完成事件。

(5)启动多个数据处理线程,负责从接收数据列表中不断的取出游戏协议数据,并调用虚函数接口ProcessMessage来处理到来的游戏协议数据。

(6)启动多个发送线程,负责从发送数据列表中取出数据,并将数据投递到IOCP上。

3.2.2　监听线程实现步骤

(1)投递一定数量的AcceptEx I/O操作,并将这些AcceptEx I/O操作记录在一个链表里。

(2)构建事件对象数组,该数组有两个成员,一个是AcceptEvent,这个事件表示投递的AcceptEx I/O已经用完了,需要重新投递一定数量的新的AcceptEx I/O操作。另一个是UsedEvent,这个事件表示一个新连接建立了,某一个AcceptEx I/O操作已经完成,需要在重新投递一个新的AcceptEx I/O操作。

(3)监听线程进入循环。

(4)调用WSAWaitForMultipleEvents函数在事件对象数组上等待事件发生,设定等待时间是60秒。

(5)如果等待超时,说明投递的AcceptEx I/O没有被完成,这时遍历未决的AccpetEx I/O链表(未决的AcceptEx I/O操作指的是已经投递出去的,但还没有返回的AcceptEx I/O操作)。在每个未决的AccpetEx I/O操作上调用getsockopt函数,检查连接建立的时间,如果超过规定的时间,则关闭这个连接,这样做可以避免恶意客户连接。回到步骤(4)继续执行。

(6)如果有事件发生,则根据不同的事件,继续投递相应数量的AcceptEx I/O操作。回到步骤(4)继续执行。

3.2.3　服务线程实现步骤

(1)服务线程进入循环,不断调用GetQueuedCompletionStatus函数来检查是否有完成的重叠I/O操作。

(2)如果没有重叠操作完成,则等待。如果有重叠操作完成,检查套接字上是否发生错误,如果有错误,进行相应的错误处理。如果没有错误,则进行相应的事件处理。

(3)如果是重叠的接收(AcceptEx I/O)操作完成,说明有新的客户连接了服务器,为新连接申请客户上下文(CClientContext)对象,并把这个新的连接关联到IOCP对象。

(4)如果是重叠的读操作完成,说明有数据到达,这时应该读取数据,读取的时候要按照顺序进行,即按照客户上下文结构中的nCurrentReadSequence变量来顺序读取。读取完成以后,将读取的数据通过NotifyReadCompleted接口函数通知用户,经过用户预处理后,把数据放进接收数据列表中。

(5)如果是重叠的写操作完成,说明数据成功发送给了客户端。将发送的数据通过NotifyWriteCompleted接口函数告知用户。

3.2.4　数据处理线程的实现步骤

(1)数据处理线程进入循环。

(2)在互斥变量的保护下访问接收数据链表,从接收数据链表中取出数据。

(3)调用虚函数接口中的ProcessMessage函数处理取到的数据。

(4)在互斥变量的保护下,将经过处理的数据放入发送数据链表中,回到步骤(2)。

3.2.5　发送线程的实现步骤

(1)发送线程启动循环。

(2)在互斥变量的保护下访问发送数据链表,从发送数据链表中取出数据。

(3)把取到的数据投递到IOCP上。如果客户方要求数据按顺序发送,这时在投递数据时要按照客户上下文结构中的nCurrentSendSequence参数来顺序投递。

(4)投递完数据后,回到步骤(2)。

3.3　留给游戏开发者的接口

(1)新客户连接建立时的处理接口,接口定义如下:

```
void NotifyNewConnection(CClientContext *pContext,CClientBuffer *pBuffer);
```

(2)客户连接关闭时的处理接口,接口定义如下:

```
void NotifyConnectionClosing (CClientContext *pContext,CClientBuffer *pBuffer);
```

(3)客户连接上发生错误时的处理接口,接口定义如下:

```
void NotifyConnectionError (CClientContext *pContext,CClientBuffer *pBuffer,int nError);
```

(4)从 IOCP 上读取到数据时的处理接口,接口定义如下:

void NotifyReadCompleted (CClientContext *pContext,CClientBuffer *pBuffer);

(5)发送数据成功时的处理接口,接口定义如下:

void NotifyWriteCompleted (CClientContext *pContext,CClientBuffer *pBuffer);

(6)处理游戏协议消息的接口,接口定义如下:

void ProcessMessage (CClientContext *pContext,CClientBuffer *pBuffer);

游戏开发者要重写这几个接口函数,完成自己的网络通信部分。其中接口函数 ProcessMessage 是必须重写的。在这里,可以调用游戏协议层解析函数,来处理到来的游戏的消息。最后,将上面的实现封装成一个底层的网络通信类 CNetServer。

4 虚函数接口的使用方法及服务器性能测试

4.1 象棋游戏服务器通信类的实现

上面开发的网络游戏底层通信类只负责底层的网络通信,并不负责有关游戏消息的处理。它留虚函数接口给游戏开发者,游戏开发者要实现这些接口,把具体的游戏消息的处理与网络通信结合起来。实现具体游戏的服务器。

以开发的象棋游戏服务器为例,来说明使用接口的方法,首先象棋游戏服务器通信类 CChessServer 继承上面开发的网络通信类 CNetServer,然后重写虚函数接口。

(1)重写新客户连接建立时的处理接口,在这个接口里面实现新客户连接建立时的处理代码。

(2)重写客户连接关闭时的处理接口,在这个接口里面实现客户关闭连接时的处理代码。

(3)重写客户连接上发生错误时的处理接口,在这个接口里面实现客户连接上发生错误时的处理代码。

(4)重写从 IOCP 上读取到数据时的处理接口,在这个接口里面可以对读到的数据进行预处理。

(5)重写处理游戏协议消息的接口,这个接口是必须要重写的,在这个接口里面调用具体游戏协议的处理方法。

4.2 游戏协议消息的处理

游戏消息处理主要是通过重写虚函数 ProcessMessage 来完成,在 ProcessMessage 中调用游戏协议层中的处理函数,来完成游戏消息的处理。

具体处理如下:

```
void ProcessMessage( CClientContext *pContext , CClientBuffer *pBuffer)
{
  //从 buff 中取到本次通信的消息内容
  Message *pMessage=(Message*)pBuffer->buff;

  //查看消息类型
  Switch(pMessage->nType)
  {
    …
  //对不同类型的消息进行处理
  Case CMD_CLIENT_LOGIN:
       ProcessLoginMessage(pMessage); //游戏协议层提供的处理函数
       Break;
  Case CMD_CLIENT_DOWN:
       ProcessDownMessage(pMessage);
       Break;
    …
  }
}
```

上面给出的是处理消息的简明代码,首先从 pBuffer 的 buff 中取到游戏消息,查看消息的类型,然后根据不同的消息类型调用游戏协议层中不同的消息处理函数来处理消息。

4.3 服务器性能测试

为了测试象棋服务器的性能,开发了一个虚拟客户端。虚拟客户端是一个多线程程序,每个线程代表一个客户。测试时虚拟客户端同时启动多个线程来连接服务器,并向服务器发送测试数据,然后等待接收经服务器处理过的数据。虚拟客户端每隔 1 秒循环做一次相同的操作。

(1)测试环境

两台双核 Genuine Intel® CPU T2080 1.73 GHz,1 024 M 内存的 PC 机。两台机器安装的操作系统都是 Windows XP。在其中一台上安装虚拟客户端,用于模拟大量的客户,另一台用于模拟服务器。

(2)测试结果及其分析

测试时,虚拟客户端分别启动了 100、200、300、400、500、600、700、800、900、1 000、1 100、1 200、1 300、1 400、1 500、1 600、1 700、1 800、1 900、2 000 个线程,代表不同数量的客户同时访问服务器。图 3 是就 CPU 平均占用率和内存消耗量两个方面对服务器测试的结果。其中左图是对 CPU 平均占用率的测试结果,横坐标是连接的客户数,纵坐标是 CPU 平均占用率(%)。右图是对内存消耗量的测试结果,横坐标是连接的客户数,纵坐标是内存消耗量(M)。

从图 3 可以看出,随着连接客户数量的增多,服务器对 CPU 的占用率并没有急速增长。而是缓慢增长,这是服务器伸缩性好的一种表现。另外,服务器对内存的消耗量也是稳定的呈线性缓慢增长,说明服务器在内存消耗上是高效的,稳定的。

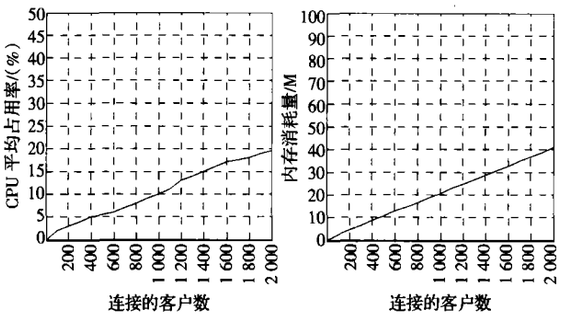

图 3 服务器测试结果

5 结束语

本文着力于完成网络游戏服务器网络通信层的实现,给出了具体的设计思路和实现方法。在此基础上开发的中国象棋网络游戏服务器性能良好,可以满足 10 个客户的同时访问的需求,也可以满足 2 000 个客户的同时访问的需求。最后就 CPU 占用率和内存消耗量两个方面对服务器进行了测试,结果表明使用 IOCP 机制开发的网络游戏服务器是高效的。

与其他 Windows 套接字 I/O 模型相比,IOCP 在管理大量

(下转 81 页)

(1)根据要编码的第 N 帧的当前宏块(x,y),计算前面第 $N-1$ 帧的所对应的 9 个宏块的平均运动能量 MEB。如果 MBE< $T1$,转到步骤(2)。否则转到步骤(3)。

(2)MEB<$T1$,在平坦背景区域的宏块内进行搜索。

(3)计算相邻四个宏块的平均运动能量 MEB,如果 MEB< $T2$,在复杂背景区的宏块内进行搜索。如果 MEB>$T2$,在运动区域的宏块内搜索。

(4)与帧内模式进行比较,选出最佳模式。

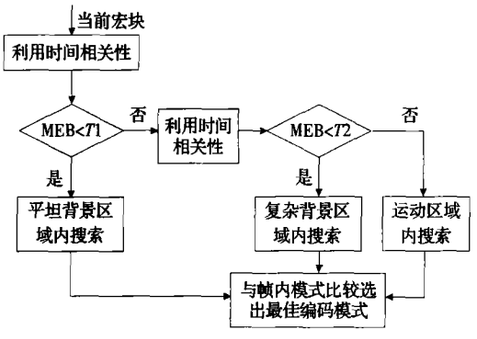

图 4　算法流程图

4　实验结果与分析

为验证该算法的性能,以 ADI 公司的 Blackfin533[11]为平台进行了测试和分析。Blackfin533 是一种微信号(MSA)的 DSP 平台。它是 16 位定点 DSP 内核,可以实现 600 MHz 的持续工作,具有专门的视频指令,有很强的运算处理能力。

测试模型的配置参数为:编码 100 帧,帧率为 30 f/s,序列编码结构为 IPPP,采用 Hadamard 变换,搜索范围为±16 像素,1 个参考帧,不使用流控,采用 CABAC。测试序列为 QCIF:Akiyo、Container、News、Saleman、Corphone、Foreman。

从实验结果表 2 中可以看出,算法与标准参考的全搜索算法相比,可以在平均信噪比降低 0.04 dB,码率基本不变的前提下平均节约 23.1%的编码时间。并且随着 QP 的增大,编码效率逐步提高,在 QP 为 28 时,编码时间只平均减少了 21.68%;而在 QP 为 36 时,编码时间平均减少了 24.51%。另一方面,在序列的运动复杂度不高的情况下,文中的算法还能在节约编码时间的同时降低比特率。例如 Akiyo 序列,在 QP 为 28 时,信噪比平均下降为 0.01 dB,编码时间平均减少了 26.4%,码率也平均下降了 0.2%。

表 2　快速算法与 H.264/AVC 算法性能比较

类别	序列	QP	△PSNR/dB	△Bitrate/(%)	△Speedup/(%)
平坦背景	Akiyo	28	-0.01	-0.2	26.4
		36	-0.02	-0.4	29.6
	Container	28	0	0.1	23.9
		36	-0.02	-0.3	26.3
复杂背景	News	28	-0.02	0.4	21.5
		36	-0.05	0.3	25.6
	Saleman	28	-0.03	0.5	22.1
		36	-0.04	0.3	24.6
运动区	Corphone	28	-0.06	0.7	18.8
		36	-0.09	0.3	20.6
	Foreman	28	-0.05	0.9	17.4
		36	-0.11	0.5	20.4

5　结束语

提出的基于运动复杂度的模式选择算法,首先根据视频图像的运动复杂程度,将视频区域划分为三种类型,计算一个区域的运动复杂度,然后根据时间和空间的相关性,减少了搜索模式。实验结果表明,在图像 PSNR 下降不足 0.04 dB 的情况下,编码速度却提升了 17%~29%,提高了编码速度。

参考文献:

[1] Draft ITU-T recommendation and final draft internationa standard joint video specification [S].(ITU-TRec.H.264‖ISO/IEC 14496-10 AVC),7th Meeting: Pattaya,Thailand,2003.

[2] Wiegand T,Sullivan G J,Bjontegaad G,et al.Overview of the H.264/AVC video coding standard[J].IEEE Trans Circuits Syst Video Technol.,2003,13:560-575.

[3] Kamaci N,Altunbasak Y.Performance comparison of the emerging H.264 video coding standard with the existing standards[C]//Pro of Multimedia and Expo,ICME'03,2003.

[4] Wu D,Wu S,Lim K P F,et al.Block inter mode decision for fas encoding of H.264[C]//Proc ICASSP,2004,3:181-184.

[5] Yu A C.Efficient block-size selection algorithm for inter-frame coding in H.264/MPEG4 AVC[C]//ICIP,2004,1:95-98.

[6] Wu D,Pan F.Fast intermode decision in H.264/AVC video coding[J].IEEE Transactions on Circuits and Systems for Video Technology,2005,15(7):953-985.

[7] 陈峰.Blackf in 系列 DSP 原理与系统设计[M].北京:电子工业出版社,2004.

(上接 78 页)

并发的客户连接请求时有着巨大的优势,而且随着访问客户的增多,这种优势更加明显,所以,对于在 Windows 平台下开发为大量客户服务的网络游戏服务器应用来说,使用 IOCP 是一种较好的选择。

参考文献:

[1] 陈和平,周静宁.IOCP 机制和网络代理服务器的实现方法[J].计算机应用,2003,23(4):109-110.

[2] Jones A,Ohlund J.Windows 网络编程[M].2 版.杨和庆,译.北京:清华大学出版社,2002.

[3] 苏羽,关沫,许研.Visual c++网络游戏建模与实现[M].2 版.北京:科学出版社,2006.

[4] 吴兆定,袁江海,郑世宝.棋牌类网络游戏服务器端架构设计[J].计算机工程,2007,33(15):283-285.

[5] 袁刚,雷光波,傅鹏,等.高性能代理服务器的实现[J].计算机工程与应用,2005,41(5):150-152.