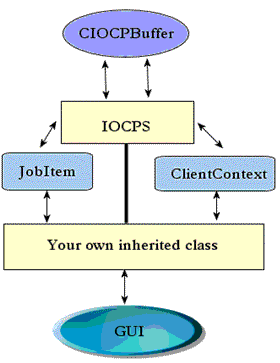


An overview of the IOCP class source code functionality. We have several IO worker threads that handle asynchronous I/O calls through the completion port (IOCP), and these workers call some virtual functions which can put requests that need a large amount of computation in a work queue. The logical workers take the job from the queue, and process it and send back the result by using some of the functions provided by the class. The Graphical User Interface (GUI) usually communicates with the main class using Windows messages (because MFC is not thread safe) and by calling functions or by using the shared variables.

 **The class overview**

* CIOCPBuffer: A class used to manage the buffers used by the asynchronous I/O calls: •Sometimes, when an asynchronous I/O call is completed, we may have partial packages in the buffer, therefore the need to split the buffer to get a complete message. This is done by the SplitBuffer function in the CIOCPS class. Also, sometimes we need to copy information between the buffer, and this is done by the AddAndFlush function in the IOCPS class
* IOCPS: The main class that handles all the communication.
* JobItem: A structure which contains the job to be performed by the logical worker threads.
* ClientContext: A structure that holds client specific information (status, data, etc.).
* Virtual functions
* NotifyNewConnection :

Called when a new connection has been established.

* NotifyNewClientContext

Called when an empty ClientContext structure is allocated.

* NotifyDisconnectedClient

Called when a client disconnects.

* ProcessJob

Called when logical workers want to process a job.

* NotifyReceivedPackage

Notifies that a new package has arrived.

* NotifyFileCompleted

Notifies that a file transfer has finished.

* Important variables

Notice that all the variables have to be exclusively locked by the function that uses the shared variables, this is important to avoid access violations and overlapping writes. All the variables with name XXX, that are needed to be locked, have a XXXLock variable.

* m\_ContextMapLock;

Holds all the client data (socket, client data, etc.).

* ContextMap m\_ContextMap;
* m\_NumberOfActiveConnections

Holds the number of connected connections.

* Important functions
* GetNumberOfConnections()

Returns the number of connections.

* CString GetHostAdress(ClientContext\* p)

Returns the host address, given a client context.

* BOOL ASendToAll(CIOCPBuffer \*pBuff);

Sends the content of the buffer to all the connected clients.

* DisconnectClient(CString sID)

Disconnects a client, given the unique identification number.

* CString GetHostIP()

Returns the local IP number.

* JobItem\* GetJob()

Removes a JobItem from the queue, returns NULL if there are no Jobs.

* BOOL AddJob(JobItem \*pJob)

Adds a Job to the queue.

* BOOL SetWorkers(int nThreads)

Sets the number of logical workers that can be called anytime.

* DisconnectAll();

Disconnect all the clients.

* ARead(…)

Makes an asynchronous read.

* ASend(…)

Makes an asynchronous send. Sends data to a client.

* ClientContext\* FindClient(CString strClient)

Finds a client given a string ID. OBS! Not thread safe!

* DisconnectClient(ClientContext\* pContext, BOOL bGraceful=FALSE);

Disconnects a client.

* DisconnectAll()

Disconnects all the connected clients.

* StartSendFile(ClientContext \*pContext)

Sends a file specified in the ClientContext structure, using the optimized transmitfile(..) function.

* PrepareReceiveFile(..)

Prepares the connection for receiving a file. When you call this function, all incoming byte streams are written to a file.

* PrepareSendFile(..)

Opens a file and sends a package containing information about the file to the remote connection. The function also disables the ASend(..) function until the file is transmitted or aborted.

* DisableSendFile(..)

Disables send file mode.

* DisableRecevideFile(..)

Disables receive file mode.

* File transfer

File transfer is done by using the Winsock 2.0 TransmitFile function. The TransmitFile function transmits file data over a connected socket handle. This function uses the operating system's cache manager to retrieve file data, and provides high-performance file data transfer over sockets. These are some important aspects of asynchronous file transferring:

* Unless the TransmitFile function is returned, no other sends or writes to the socket should be performed because this will corrupt the file. Therefore, all the calls to ASend will be disabled after the PrepareSendFile(..) function.
* Since the operating system reads the file data sequentially, you can improve caching performance by opening the file handle with FILE\_FLAG\_SEQUENTIAL\_SCAN.
* We are using the kernel asynchronous procedure calls while sending the file (TF\_USE\_KERNEL\_APC). Use of TF\_USE\_KERNEL\_APC can deliver significant performance benefits. It is possible (though unlikely), however, that the thread in which the context TransmitFile is initiated is being used for heavy computations; this situation may prevent APCs from launching.

The file transfer is made in this order: the sever initializes the file transfer by calling the PrepareSendFile(..) function. When the client receives the information about the file, it prepares for it by calling the PrepareReceiveFile(..), and sends a package to the sever to start the file transfer. When the package arrives at the server side, the server calls the StartSendFile(..) function that uses the high performance TransmitFile function to transmit the specified file.

* The source code example

The provided source code example is an echo client/server that also supports file transmission (figure 4). In the source code, a class MyIOCP inherited from IOCP handles the communication between the client and the server, by using the virtual functions.The most important part of the client or server code is the virtual function **NotifyReceivedPackage**. The function handles an incoming message and performs the request sent by the remote connection. In this case, it is only a matter of a simple echo or file transfer. The source code is divided into two projects, IOCP and IOCPClient, which are the server and the client side of the connection.

**IOCPS::Startup()函数分析**

* 调用CreateCompletionPort()函数创建完成端口
* 调用SetupListner()完成监听线程的启动:调用如下的线程启动函数，实现ListnerThreadProc线程的启动.AfxBeginThread(IOCPS::ListnerThreadProc, (void\*)this,THREAD\_PRIORITY\_NORMAL);对于监听线程来说，只要有连接请求到来并判定是FD\_ACCEPT事件，则执行WSAAccept函数，函数接受连接请求后会得到一个clientSocket，然后将这个clientSocket通过调用函数AssociateIncomingClientWithContext将新连接进来的clientSocket与客户端上下文绑定.AssociateIncomingClientWithContext函数内部首先是调用了如下一句代码：ClientContext\* pContext = AllocateContext(); //分配一个客户端上下文；实现了一个客户端上下文环境变量的创建，如果创建成功，则用上面新连接进来的clientSocket套接字来初始化新创建的pContext变量.从而实现上下关联。上面的关联完成后还调用了setsockopt函数对套接字clientSocket作了设置接着还调用了函数AddClientContext(pContext)将上面的上下文环境添加到了上下文环境映射表当中.
* 调用函数AssociateSocketWithCompletionPort(clientSocket, m\_hCompletionPort, (DWORD) pContext)将上面的clientSocket与第二步中用CreateCompletionPort()函数创建完成端口绑定。然后调用AllocateBuffer(IOInitialize)为新连接进来的clientSocket创建一个CIOCPBuffer数据用来通讯然后调用PostQueuedCompletionStatus(m\_hCompletionPort, 0, (DWORD) pContext, &pOverlapBuff->m\_ol)， 向完成端口投递一个状态；调用SetupIOWorkers()，函数实现工作线程的启动。在工作线程启动函数调用AfxBeginThread(IOCPS::IOWorkerThreadProc, (void\*)this,THREAD\_PRIORITY\_NORMAL);创建了一个线程，并将创建的工作线程添加到m\_IOWorkerList链表当中m\_IOWorkerList.AddHead((void\*)pWorkerThread);

(对于工作线程的工作描述：主要是通过

// Get a completed IO request.

BOOL bIORet = GetQueuedCompletionStatus(

hCompletionPort,

&dwIoSize,

(LPDWORD) &lpClientContext,

&lpOverlapped, INFINITE);

阻塞函数等等完成端口的状态变化。)

同时获取CIOCPBuffer \*pOverlapBuff;利用下面的一句代码得到了接收到的数据.

pOverlapBuff=CONTAINING\_RECORD(lpOverlapped, CIOCPBuffer, m\_ol);

同时还调用了函数ProcessIOMessage(pOverlapBuff, lpClientContext, dwIoSize);

来进行内部消息判定是什么操作，然后通过相应的switch...case语句，执行相应的

IOxxx函数，列表如下：

// Used by IO Workers.

OnWriteCompleted(ClientContext \*pContext, DWORD dwIoSize,CIOCPBuffer \*pOverlapBuff);

OnWrite(ClientContext \*pContext, DWORD dwIoSize,CIOCPBuffer \*pOverlapBuff);

OnReadCompleted(ClientContext \*pContext, DWORD dwIoSize,CIOCPBuffer \*pOverlapBuff=NULL);

OnRead(ClientContext \*pContext,CIOCPBuffer \*pOverlapBuff=NULL);

OnInitialize(ClientContext\* pContext, DWORD dwIoSize,CIOCPBuffer \*pOverlapBuff=NULL);

OnPostedPackage(ClientContext \*pContext,CIOCPBuffer \*pOverlapBuff);.

* 创建逻辑工作线程池SetWorkers(m\_nOfWorkers);
* PostQueuedCompletionStatus函数，向每个工作者线程都发送—个特殊的完成数据包。该函数会指示每个线程都“立即结束并退出”.下面是PostQueuedCompletionStatus函数的定义：

BOOL PostQueuedCompletionStatus(

HANDLE CompletlonPort,

DW0RD dwNumberOfBytesTrlansferred,

DWORD dwCompletlonKey,

LPOVERLAPPED lpoverlapped,

);

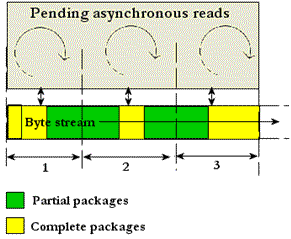
其中，CompletionPort参数指定想向其发送一个完成数据包的完成端口对象。而就dwNumberOfBytesTransferred,dwCompletionKey和lpOverlapped这三个参数来说.每—个都允许我们指定—个值,直接传递给GetQueuedCompletionStatus函数中对应的参数。这样—来。—个工作者线程收到传递过来的三个GetQueuedCompletionStatus函数参数后，便可根据由这三个参数的某一个设置的特殊值，决定何时应该退出。例如,可用dwCompletionPort参数传递0值,而—个工作者线程会将其解释成中止指令。一旦所有工作者线程都已关闭,便可使用CloseHandle函数,关闭完成端口。最终安全退出程序。

PostQueuedCompletionStatus函数提供了一种方式来与线程池中的所有线程进行通信。如，当用户终止服务应用程序时，我们想要所有线程都完全利索地退出。但是如果各线程还在等待完成端口而又没有已完成的I/O 请求，那么它们将无法被唤醒。

通过为线程池中的每个线程都调用一次PostQueuedCompletionStatus，我们可以将它们都唤醒。每个线程会对GetQueuedCompletionStatus的返回值进行检查，如果发现应用程序正在终止，那么它们就可以进行清理工作并正常地退出。

PostQueuedCompletionStatus主要是投递一个任务到完成队列当中,从而使得在等待队列消息的某一个线程收取到.其参与分别与GetQueuedCompletionStauts相对应,从而可以很方便地为在等待完成消息的线程(池)分派任务,而不需要另外再开线程资源.基于这一种特性,还可以把完成端口当成一个高效的队列+线程池.如果你是想退出线程的话,也可以通过这种方式投递特定的消息.由于退出消息一个线程只会处理一个(这个逻辑问题应该不用解释),所以如果想让所有业务线程退出,就只需要根据线程数量投递多个退出消息即可.

* The WSAENOBUFS error problem
* Ways : We perform an asynchronous WSARead(..) (see OnZeroByteRead(..)) with a zero byte buffer. When this call completes, we know that there is data in the TCP/IP stack, and we read it by performing several asynchronous WSARead(..) with a buffer of MAXIMUMPACKAGESIZE. This solution locks physical memory only when data arrives, and solves the WSAENOBUFS problem. But this solution decreases the throughput of the server
* The package reordering problem: This problem is also being discussed by [3]. Although committed operations using the IO completion port will always be completed in the order they were submitted, thread scheduling issues may mean that the actual work associated with the completion is processed in an undefined order. For example, if you have two I/O worker threads and you should receive "byte chunk 1, byte chunk 2, byte chunk 3", you may process the byte chunks in the wrong order, namely, "byte chunk 2, byte chunk 1, byte chunk 3". This also means that when you are sending the data by posting a send request on the I/O completion port, the data can actually be sent in a reordered way.
* Ways: A simple practical solution to this problem is to add a sequence number to our buffer class, and process the data in the buffer if the buffer sequence number is in order. This means that the buffers that have incorrect numbers have to be saved for later use, and because of performance reasons, we will save the buffers in a hash map object (e.g., m\_SendBufferMap and m\_ReadBufferMap) GetNextSendBuffer and GetNextReadBuffer, to get the ordered send or receive buffer. IncreaseReadSequenceNumber and IncreaseSendSequenceNumber, to increase the sequence numbers.
* Asynchronous pending reads and byte chunk package processing problem
* Ways: if we want to use the IOCP server's full potential, we should have several pending asynchronous reads waiting for data to arrive. This means that several asynchronous reads complete out of order (as discussed before in section 3.6.2), and byte chunk streams returned by the pending reads will not be processed in order. Furthermore, a byte chunk stream can contain one or several packages and also partial packages. This means that we have to process the byte stream chunks in order to successfully read a complete package. Furthermore, we have to handle partial packages (marked with green in figure 1). This makes the byte chunk package processing more difficult. The full solution to this problem can be found in the ProcessPackage(..) function in the IOCPS class.



* The access violation problem: This is a minor problem, and is a result of the design of the code, rather than an IOCP specific problem. Suppose that a client connection is lost and an I/O call returns with an error flag, then we know that the client is gone. In the parameter CompletionKey, we pass a pointer to a structure ClientContext that contains client specific data. What happens if we free the memory occupied by this ClientContext structure, and some other I/O call performed by the same client returns with an error code, and we transform the parameter CompletionKey variable of DWORD to a pointer to ClientContext, and try to access or delete it? An access violation occurs!
* Ways: The solution to this problem is to add a number to the structures that contain the number of pending I/O calls (m\_nNumberOfPendlingIO), and we delete the structure when we know that there are no more pending I/O calls. This is done by the EnterIoLoop(..) function and ReleaseClientContext(..).

Rule of thumb #1:

Never read/write to the client context (e.g., ClientContext) without locking it using the context lock as in the example below. The notification function (e.g., Notify\*(ClientContext \*pContext)) is already “thread safe”, and you can access the members of ClientContext without locking and unlocking the context.

//Do not do it in this way

// …

If(pContext->m\_bSomeData)

pContext->m\_iSomeData=0;

// …

// Do it in this way.

//….

pContext->m\_ContextLock.Lock();

If(pContext->m\_bSomeData)

pContext->m\_iSomeData=0;

pContext->m\_ContextLock.Unlock();

//…

Also, be aware that when you are locking a Context, other threads or GUI would be waiting for it.

Rule of thumb #2:

Avoid or "use with special care" code that has complicated "context locks" or other types of locks inside a “context lock”, because this may lead to a “deadlock” (e.g., A waiting for B that is waiting for C that is waiting for A => deadlock).

pContext-> m\_ContextLock.Lock();

//… code code ..

pContext2-> m\_ContextLock.Lock();

// code code..

pContext2-> m\_ContextLock.Unlock();

// code code..

pContext-> m\_ContextLock.Unlock();

The code above may cause a deadlock.

Rule of thumb #3:

Never access a client context outside the notification functions (e.g., Notify\*(ClientContext \*pContext)). If you do, you have to enclose it with m\_ContextMapLock.Lock(); … m\_ContextMapLock.Unlock();. See the source code below.

ClientContext\* pContext=NULL ;

m\_ContextMapLock.Lock();

pContext = FindClient(ClientID);

// safe to access pContext, if it is not NULL and are Locked (Rule of thumbs#1:) code .. code..

m\_ContextMapLock.Unlock();

// Here pContext can suddenly disappear because of disconnect. // do not access pContext members here.

* ProcessIOMessage:读写文件已经读写文件完成消息
* IOInitialize

BOOL IOCPS::AssociateIncomingClientWithContext(SOCKET clientSocket)

CIOCPBuffer \*pOverlapBuff =AllocateBuffer(IOInitialize);

* IOReadCompleted

void IOCPS::OnRead(ClientContext \*pContext,CIOCPBuffer \*pOverlapBuff)

pOverlapBuff=AllocateBuffer(IOReadCompleted);

pOverlapBuff->SetOperation(IOReadCompleted);

* IO\_READ

BOOL IOCPS::ARead(ClientContext \*pContext,CIOCPBuffer \*pOverlapBuff)：pOverlapBuff=AllocateBuffer(IORead) (CIOCPBuffer pBuff->SetOperation(nType);

); pOverlapBuff->SetOperation(IORead);

* IO\_WRITE

BOOL IOCPS::ASendToAll(CIOCPBuffer \*pBuff): CIOCPBuffer \*pOverlapBuff=AllocateBuffer(IOWrite);

BOOL IOCPS::PrepareSendFile(ClientContext \*pContext, LPCTSTR lpszFilename): CIOCPBuffer \*pOverlapBuff=AllocateBuffer(IOWrite);

BOOL IOCPS::StartSendFile(ClientContext \*pContext)

CIOCPBuffer \*pOverlapBuff=AllocateBuffer(IOWrite);

void MyIOCP::PackageFileTransfer(CIOCPBuffer \*pOverlapBuff,int nSize,ClientContext \*pContext) CIOCPBuffer \*pOverlapBuff=AllocateBuffer(IOWrite);

CIOCPBuffer \*pOverlapBuff=AllocateBuffer(IOWrite);

BOOL MyIOCP::BuildPackageAndSend(ClientContext \*pContext, CString sText)

CIOCPBuffer \*pOverlapBuff=AllocateBuffer(IOWrite);

BOOL MyIOCP::BuildPackageAndSendToAll(CString sText)

CIOCPBuffer \*pOverlapBuff=AllocateBuffer(IOWrite);

BOOL MyIOCP::BuildStartFileTransferPackageAndSend(ClientContext \*pContext)

CIOCPBuffer \*pOverlapBuff=AllocateBuffer(IOWrite);

* IOWriteCompleted

void IOCPS::OnWrite(ClientContext \*pContext, DWORD dwIoSize,CIOCPBuffer \*pOverlapBuff)

pOverlapBuff->SetOperation(IOWriteCompleted);

* IOZeroByteRead

BOOL IOCPS::AZeroByteRead(ClientContext \*pContext, CIOCPBuffer \*pOverlapBuff)

pOverlapBuff=AllocateBuffer(IOZeroByteRead); pOverlapBuff->SetOperation(IOZeroByteRead);

* IOZeroReadCompleted

void IOCPS::OnZeroByteRead(ClientContext \*pContext,CIOCPBuffer \*pOverlapBuff)

pOverlapBuff=AllocateBuffer(IOZeroReadCompleted);

pOverlapBuff->SetOperation(IOZeroReadCompleted);

BOOL IOCPS::StartSendFile(ClientContext \*pContext)

pOverlapBuff->SetOperation(IOTransmitFileCompleted);

* IOPostedPackage

BOOL IOCPS::PostPackage(ClientContext \*pContext,CIOCPBuffer \*pOverlapBuff)

pOverlapBuff->SetOperation(IOPostedPackage);

* SetupListner启动IOCPS::ListnerThreadProc线程；SetupIOWorkers启动IOCPS::IOWorkerThreadProc线程；SetWorkers （Sets the number of Workers (NOT IOWorkers that deals with Send/Receive，logical worker thread pool to process heavier client/server requests or computations)启动IOCPS::WorkersThreadProc线程

1. IOCPS::ListnerThreadProc线程：监听是否有新的连接请求
2. IOCPS::IOWorkerThreadProc线程：处理数据的接受、发送
3. IOCPS::WorkersThreadProc线程： Start the logical Workers. (SetWorkes can be called in runtime..)：

* AssociateIncomingClientWithContext

/\*

\* AssociateIncomingClientWithContext

\* This function do the Following:

\* 1) Does some simple Secutity Stuff (e.g one connection per client, etc..)

\* 2) Allocates an Context for the Socket.

\* 3) Configures the Socket.

\* 4) Associate the Socket and the context with the completion port.

\* 5) Fires an IOInitialize So the IOWORKERS Start to work on the connection.

\*/



* #define SIMPLESECURITY（IOCPS.h）：Add this if you want to be able to block certain IP address or just allow one connection per IP
* ARead why not directly to Call WSARecv??：To uniformely distribute the CPU power among all the clients/work packages in the IOCP queue.
* Q1: IOCPS::Startup中if ( m\_iNumberOfPendlingReads>1&&m\_iMaxIOWorkers>1 )这个判断条件永远不会成立，因为前面的代码：if ( m\_iMaxIOWorkers>1 ) m\_iNumberOfPendlingReads=1;（// we have some sort of bug somewhere.）
* MyIOCP::ProcessJob(virtual inline)：This functions defines what should be done when a job from the work queue is arrived (not used). IOCPS::ProcessJob(virtual inline): Virtual Function who Processes a Job. Used to do rare heavy computation or calls that blocks the calling thread for a while.
* **以下**<http://www.codeproject.com/Articles/10330/A-simple-IOCP-Server-Client-Class中评论内容（08>年以后**）**

1. Is there a fatal mistake in this sample?

Thank the author for his excellent work of the code!

But I think there is a fatal mistake in it. Please take a look at 3.6.2 about the packet reorder. "Although committed operations using the IO completion port will always be completed in the order they were submitted, thread scheduling issues may mean that the actual work associated with the completion is processed in an undefined order." That's TRUE. "A simple practical solution to this problem is to add a sequence number to our buffer class, and process the data in the buffer if the buffer sequence number is in order. This means that the buffers that have incorrect numbers have to be saved for later use, and because of performance reasons, we will save the buffers in a hash map object (e.g., m\_SendBufferMap and m\_ReadBufferMap)."

It's also an excellent solution to resolve this problem. But in the code, let's see the solution:

void CIOCPS::MakeOrderdRead(ClientContext \*pContext, CIOCPBuffer \*pBuff) {

if(pContext != NULL && pBuff != NULL) {

//pContext->m\_ContextLock.Lock();

**pBuff->SetSequenceNumber(pContext>m\_ReadSequenceNumber); // this line**

//TRACE("MakeOrderdRead> %i\r\n", pBuff->GetSequenceNumber());

DWORD dwIoSize = 0; ULONG ulFlags = MSG\_PARTIAL; UINT nRetVal = WSARecv(pContext->m\_Socket, pBuff->GetWSABuffer(), 1, &dwIoSize, &ulFlags, &pBuff->m\_ol, NULL);

if(nRetVal == SOCKET\_ERROR && WSAGetLastError() != WSA\_IO\_PENDING) {

if(WSAGetLastError() != WSAENOTSOCK) { CString msg;

msg.Format("Disconnect in Onread Possible Socket Error: %s", WSAErrCode2Text(WSAGetLastError()));

SaveLog(msg); }

//pContext->m\_ContextLock.Unlock();

ReleaseBuffer(pBuff);

TRACE(">MakeOrderdRead(%x)\r\n", pContext);

DisconnectClient(pContext);

ReleaseClientContext(pContext); }

else { pContext->m\_ReadSequenceNumber = (pContext->m\_ReadSequenceNumber + 1) % MAXIMUMSEQUENSENUMBER;

//pContext->m\_ContextLock.Unlock(); } } }

Take a look at the bold line text.

pBuff->SetSequenceNumber(pContext>m\_ReadSequenceNumber);

If current IOWorkerThread()(ThreadA)this line execute finish and another IOWorkerThread() get the CPU,and it also run at this line and it WSARecv() immediately, and then ThreadA continue running, and call WSARecv().As you see, the Recv MSG post in incorrect order, so IOCP awake IOWorkerThread in incorrect order.

So, the code should still insert zhe line "//pContext-&gt;m\_ContextLock.Lock();" and I dont know why the author cut this from the function. The same mistake also in BOOL CIOCPBase::ASend(ClientContext \*pContext, CIOCPBuffer \*pOverlapBuff) function. Always we can't guarantee that the IOCPS->SetSendSequenceNumber() function and WSASend() function execute togehter atomic. Or can't guarantee IOCPBuffer->SetSequenceNumber() function and WSARecv() function execute togehter atomic. Am I think in wrong way? Is there someone can discuss the question with me?

**作者**：Hello，Good analysis and call.

It was a while ago I wrote this code. I see that I have removed the Context lock(). The reason for this as I recall is the following:

"For each context (or socket) you make one call to WSARecv(..) at the time, independent on number of working Threads. IOCP is thread safe (basically a queue of finished async calls) and since you make only one WSARecv(..) per client at the time you will not have this. This would not work if you have several pending reads for the same Socket. (see Multiple pending read removed when multiple I/O workers are used. (Temporary fix is now permanent fix, read A6 and Q6.) in the article). " I removed that since the benefit of having several pending reads or buffers in kernel (TCP/IP) waiting for data since you have to shuffle them around or implement locked sequential buffer shuffling that is not good. The reason of sequencing is because of CPU Load sharing (that the code do itself) see the function ASend(..) to understand. ( I make a post into the Queue so that the CPU is shared fairly among the client independent on the their internet connection.) "

However my memory can make tricks on me (or windows API) so I ask you:

Did you experimentally see this issue or is it just a code analysis?

1. A bug in the MyIOCP::BuildPackageAndSend(int ClientID, CString sText)

First thanks for sharing this article and I learned a lot IOCP from you.

BOOL MyIOCP::BuildPackageAndSend(int ClientID, CString sText)

{

BOOL bRet=FALSE;

m\_ContextMapLock.Lock();

ClientContext\* pContext = FindClient(ClientID);

if (pContext == NULL)

{

m\_ContextMapLock.Unlock(); //this line should be added here

return FALSE;

}

bRet=BuildPackageAndSend(pContext,sText);

m\_ContextMapLock.Unlock();

return bRet;

}

In my experiment, without this line the non-paged memery goes up very quickly in some occassions, I remember someone said the computer stopped working, this may be the reason. there are so many messages here, if someone has mentioned this, sorry for wasting your time reading this.

//以下是源代码作者的回复

You are absolutely right. I wonder if this is in the code uploaded in the code project already? Laugh | :laugh: If not I should put it there. PS. MyIOCP was only a 20 min demo wrapper so it can contain bugs etc. it is IOCPS.\* that contain the important stuff.

Many thanks anyway.

1. Is that a bug?

In the function of AddToFile, when the server received other data which is not the file,the file corrupted,it should be closed, otherwise the server will close the connect. In debug ,it will received thus infos: "BUffer after pBuffFileRemain"

"Context a9cdc48 is is disconnected but not removed from Context map "

void IOCPS::AddToFile(ClientContext \*pContext, DWORD dwIoSize, CIOCPBuffer \*pOverlapBuff)

{ if (pContext->m\_File.m\_hFile != NULL)

{ pContext->m\_ContextLock.Lock();

int iBytesLeft=(int)pContext->m\_iMaxFileBytes-pContext->m\_iFileBytes;

// The buffer contains only data to be written to the buffer.

if(iBytesLeft>=dwIoSize)

{ pContext->m\_File.Write(pOverlapBuff->GetBuffer(), dwIoSize );

pContext->m\_iFileBytes+=dwIoSize;

// We are finished.

if(pContext->m\_iFileBytes==pContext->m\_iMaxFileBytes)

{ NotifyFileCompleted(pContext);

pContext->m\_File.Close();

pContext->m\_bFileReceivedMode=FALSE;

}

ReleaseBuffer(pOverlapBuff);

}else { PBuffer \*pBuffFileRemain=AllocateBuffer(0);

if(pBuffFileRemain!=NULL)

{AddAndFlush(pOverlapBuff,pBuffFileRemain,iBytesLeft);

TRACE("BUffer after pBuffFileRemain>\r\n");

pContext->m\_File.Write(pBuffFileRemain->GetBuffer(), iBytesLeft );

pContext->m\_iFileBytes+=iBytesLeft;

ReleaseBuffer(pBuffFileRemain);

if(pContext->m\_iFileBytes==pContext->m\_iMaxFileBytes)

{ NotifyFileCompleted(pContext);

pContext->m\_File.Close();

pContext->m\_bFileReceivedMode=FALSE; }

//The file should be closed,otherwise the server will closed client

else{

pContext->m\_File.Close();

pContext->m\_bFileReceivedMode=FALSE; }

ProcessPackage(pContext,dwIoSize-iBytesLeft,pOverlapBuff);

}else { pContext->m\_ContextLock.Unlock();

AppendLog("Error could not allocate buffer");

ReleaseBuffer(pOverlapBuff);

DisconnectClient(pContext);

return;

} }

pContext->m\_ContextLock.Unlock(); } }

作者回复：Yes, It is a bug, I think. But I am not sure of the closing of file part. Can you explain more?

问题作者：Sorry,I can't explain more.I just know if i don't close the file, the server will close the client. If the file is not closed, the server will closed the client, and at the same time, it will push the other data that the client sended, but of the file. Other wise, when the client sends the file data,it also sends other data in the other thread, I am not sure the case. Sorry,I am pool english.

1. Can I change MAXIMUMPACKAGESIZE to 1024

Depends on your page size. I think I wrote some comments in the code or in the article about this.

1. Question here regarding function IOWorkerThreadProc

hi, i have a question here regarding function IOWorkerThreadProc

// If Something whent wrong..

if (!bIORet)

{

// Clear the buffer if returned.

pOverlapBuff=NULL;

if(lpOverlapped!=NULL)

pOverlapBuff=CONTAINING\_RECORD(lpOverlapped, CIOCPBuffer, m\_ol);

if(pOverlapBuff!=NULL)

pThis->ReleaseBuffer(pOverlapBuff);

continue;

}

my question is if pOverlapBuff=NULL, when come to pThis->ReleaseBuffer(pOverlapBuff);

POSITION pos=pBuff->GetPosition();

i found that pos will be 0xcccccccc which will cause assertion. is it a bug? or my thinking is wrong? Thanks for your reply

作者:(未确定这个bug)

Sorry for the late response. I do not recognize this code. Can you please send me the full code or explain more. I do not understand the Question.

1. Fix bug:作者未回复

void IOCPS::OnReadCompleted(ClientContext \*pContext, DWORD dwIoSize,CIOCPBuffer \*pOverlapBuff)

if(m\_bReadInOrder)

pOverlapBuff=GetNextReadBuffer(pContext,pOverlapBuff);

{while(pOverlapBuff!=NULL)

{ //TRACE("R> %i\r\n",pOverlapBuff->GetSequenceNumber());

// Mark that we are Using the buffer..

pOverlapBuff->Use(dwIoSize);

#ifdef TRANSFERFILEFUNCTIONALITY

if(!pContext->m\_bFileReceivedMode)

#endif

ProcessPackage(pContext,dwIoSize,pOverlapBuff);

#ifdef TRANSFERFILEFUNCTIONALITY

else

AddToFile(pContext,dwIoSize,pOverlapBuff);

#endif

IncreaseReadSequenceNumber(pContext);

pOverlapBuff=NULL;

if(m\_bReadInOrder)

pOverlapBuff=GetNextReadBuffer(pContext);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

pOverlapBuff will be changed in FUN[GetNextReadBuffer], so dwIOSize must be set at first; IOCP != FIFO. Nemo.Stone [20090527]

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

FIX：

pOverlapBuff->Use(dwIoSize);

if(m\_bReadInOrder)

pOverlapBuff=GetNextReadBuffer(pContext,pOverlapBuff);

while(pOverlapBuff!=NULL)

{

ProcessPackage(pContext,pOverlapBuff->GetUsed(),pOverlapBuff);

IncreaseReadSequenceNumber(pContext);

pOverlapBuff=NULL;

if(m\_bReadInOrder)

pOverlapBuff=GetNextReadBuffer(pContext);

}

1. **Beautiful Code. But there is a BUG in IOCPS.CPP(1733)**

Amin Gholiha:

Beautiful code And it is Very useful for me . But I find a bug; If the server exit and invoke the function ShutDown(),line 1733 “closesocket(m\_socListen);” can not send a Socket Message. So thread ListnerThreadProc line 48 "WSAWaitForMultipleEvents" can not be actived.After main thread exit,thread ListnerThreadProc dead,even WSAWaitForMultipleEvents is not be executed.

We can find an error like underside:Detected memory leaks! Dumping objects ->

thrdcore.cpp(166) : {82} client block at 0x003D63B0, subtype 0, 112 bytes long.

a CWinThread object at $003D63B0, 112 bytes long Object dump complete.

Thanks

zhq

boyfriend\_1980@163.com

作者：Yes you are right. We should close the listening thread before closing the socket. But as far as I know the WSAWaitForMultipleEvents should react since it is bounded to the socket that is now closed (according to Windows API). But sometimes things does not work as they are documented.

其他人：

DWORD code;

BOOL res = GetExitCodeThread(m\_pListenThread->m\_hThread, &code);

if (!res && code==STILL\_ACTIVE)//

{

TRACE(\_T("listenthread still alive\n"));

}

how to solve this problem, sometimes the memory leak, sometimes not leak.

I check the listenthread,but still alive.

void IOCPS::ShutDown()

......

WSASetEvent(m\_hEvent);

if(m\_nPortNumber>0)

{

AppendLog("Closing listner thread.");

WSACloseEvent(m\_hEvent);

LINGER lingerStruct;

lingerStruct.l\_onoff = 1;

lingerStruct.l\_linger = 0;

setsockopt(m\_socListen, SOL\_SOCKET, SO\_LINGER,

(char \*)&lingerStruct, sizeof(lingerStruct) );

CancelIo((HANDLE) m\_socListen);

closesocket(m\_socListen);

}

1. m\_iMaxIOWorkers and Processors in SYSTEM

Very nice code and useful code. I have few questions.

1. In some articles i read that number of IO workers thread is = Number of processor\*2 but u stress using one thread per processor is better option why? i can't understand.

2. The concurrency value shows number of concurrent threads used by kernal to manage intern IOCP related stuff not related to our IO workers thread isn't it? What is the exact relation between concurrency value and our Worker thread.

Please reply ASAP. I am realy confuse.

作者：1) Depends on the CPU. Multiple core => Several threads. Hyper threding => Several Threads. If you are developing a server, you need to have test code that stress the server + a very good profiler. (because of many issues, debugging, performance, memory leakage). This gives you indication on where you spend cpu time and what is the best configuration for a specific system. You should choose the setting that gives you the best result.

2) As far as I know, IOCP wakes up it's worker threads using LIFO logic. If concurrency value of an IOCP is 1 then only one thread can run at any time but if it's execution is blocked by any reason IOCP wakes up another thread and concurrency is preserved. If formerly blocked thread wakes up while latter thread executes then it may exist 2 threads concurrently executing for a while. So number of IO should be equal to concurrently thread. However from practical experience of the implementation of IOCP API (Microsoft) the value of this parameter is not that important.

8． ReleaseClientContext(pContext) problem：未解决

// Create the Client context to be associted with the completion port

ClientContext\* pContext = AllocateContext();

if(pContext!=NULL) {

pContext->m\_Socket = clientSocket;

// M\_ID is Used for Delete(). Should we remove this soon ?

pContext->m\_ID=clientSocket;

/\* \* TCP\_NODELAY BOOL=TRUE Disables the "nagle algorithm for send coalescing" which delays short packets in the hope that the application will send more data and allow it to combine them into a single one to improve network efficiency. \*/

const char chOpt = 1;

int nErr = setsockopt(pContext->m\_Socket, IPPROTO\_TCP, TCP\_NODELAY, &chOpt, sizeof(char));

if (nErr == -1) {

CString msg;

msg.Format("setsockopt() error: %s",ErrorCode2Text(WSAGetLastError()));

AppendLog(msg);

TRACE(">AcceptIncomingClient(%x)\r\n",pContext);

**ReleaseClientContext(pContext); Sniff | :^) // This is a problem**

return FALSE;

}

1. prupose of Worker threads

I have downloaded the sample and try to understand the logic u made. The Concept of IOWorker thread i got. But what is the purpose of workerThreads as in this sample no workerthread is running to process any job?.

Second IO worker threads can execute only one job at a time. However we can post as many request as we want aysnchronously to the completion port. The I/O worker thread will not get next post until it has Executed the one in the queue. For example it is writing some thing for clent1 until the CIOBufefr in not empty it will not pick the next post, mean while client2 request some data from server. Is this delay? How can we avoid this delay if we have more then 1000 client requests in the queue. The last client in the queue have to wait a long time. What u said about it?

作者：The reason for the workers is to allow heavier computation on other threads than the IOWorkers. It is correct that the IOWorkers can execute one job at the time but remember that we use asynchronous calls with IOCP, this means that an IOworker does not need to wait for a send/receive to be completed and can handle many simultaneous clients at the same time. However it is important that IOworkers does not get occupied with other work than Input/Output (hence the normal workers). This implementation makes it possible to handle many connections at the same time with only one thread (See the demo) The formal way to create a proper IOCP protocol is to split the task into smaller ones and combine this IOCP (post other package types into IOCP representing state change) technology with the “finite state machine” to obtain a fair CPU distribution and use several IOWorkers.

1. WSAENOBUFS on WSASend

Hi, Very well written article.

Apparently, there is another case where I'm seeing the WSAENOBUF error. When I do an Overlapped WSASend (using I/O Completion ports) with a large number of wsabufs (on my system, anything greater than 2048) seems to return this error. Now, I initially thought that this was dependent on the "amount of data" that I'm sending into WSASend. So, I ran a quick experiment.

I invoked WSASend with wsabuf count > 2048 and each buffer had 512 bytes of data. It failed and returned this error. Now, I ran the same test with each buffer having only 16 bytes of data and it failed again. So, it appears that this error is dependent on the count of buffers than on the amount of data passed into them.

Any thoughts on this? Have you guys seen similar behavior?

作者：Thanks。What you have been written is analogous to Q6 &amp; A6 in the article.

The reason why error is depended on the count of buffers is the following:

When the OS reserves memory for asynchronous calls (e.g. WSASend) it has to allocate (Read about virtual memory) a complete page block size (dependent on the system usually 1024 or 512 bytes).

The page size can be obtained by:

SYSTEM\_INFO sSysInfo; // Useful information about the system

GetSystemInfo(&sSysInfo); // Initialize the structure.

printf ("This computer has page size %d.\n", sSysInfo.dwPageSize);//当前电脑为4096

The memory that OS has reserved for asynchronous calls can not be paged out of memory (because they can be used at any given random moment) and when the maximum amount of non-paged pool (~1/4 of the physical memory) has been reached the WSAENOBUFS error will happened.

My recommendation is to perform only one WSASend call at the time when it is finished and you get the notification you can send the next chunk of data (should be the same as sSysInfo.dwPageSize to use the Hardware architecture in efficient way).

This is how this is implemented in the code.

You can also try:

BOOL PASCAL TransmitPackets(

SOCKET hSocket,

LPTRANSMIT\_PACKETS\_ELEMENT lpPacketArray,

DWORD nElementCount,

DWORD nSendSize,

LPOVERLAPPED lpOverlapped,

DWORD dwFlags

);

1. m\_iMaxIOWorkers， blocks file transfer

I am testing code with several configurations, transferring files many times to see if it is stable or not... When I change the m\_iMaxIOWorkers>1, for example 10, the transfer blocks in some stage of files transfer. With m\_iMaxIOWorkers=1, I could transfer thousands of documents with no problem.

作者：

Sounds like you have a deadlock in your protocol that you apply on top of IOCPS.h class. I have never experienced this (not even on a dual core mashine) Do you get this when you run the demo example or your own code?

Read: 7 Special considerations & rule of thumbs

Be careful of the m\_iMaxIOworker. When you do a transmit file a low level kernel thread is used to send all of the data but if you occupy to many threads with asynchronously read (e.g. m\_iMaxIOworker > 10) you will get some problems. The m\_iMaxIOworker should be the same number as number of CPU (or core). You can acctually assign the threads to different CPU or core. (not implemented in this example)

1. MAXIMUMPACKAGESIZE

if you define it like

#define MAXIMUMPACKAGESIZE 10240

then you send a 120 bytes packet , the code will allocate 10240 bytes for the 120 bytes packet. so you waste your memory

作者：Yes this is correct as I have written several times, in previous questions about sending big buffers or messages. The MAXIMUMPACKAGESIZE is directly linked to the virtual page size of the system. A big MAXIMUMPACKAGESIZE will consume a low of memory per client. This value must be chosen wisely to be multible of the page size of the system usually (1024 or 512). To send a large amount of data you need to split it into MAXIMUMPACKAGESIZE chunks and send each segment separately

1. 其他问题

第1个问题发生在IOCPS.cpp文件的void IOCPS::ShutDownIOWorkers()方法。

void IOCPS::ShutDownIOWorkers()

{

DWORD dwExitCode;

m\_bShutDown=TRUE;

// Should wait for All IOWorkers to Shutdown..

BOOL bIOWorkersRunning=TRUE;

CWinThread\* pThread=NULL;

while(bIOWorkersRunning)

{

// Send Empty Message into CompletionPort so that the threads die.

if(bIOWorkersRunning) //如果bIOWorkersRunning始终为TRUE，那么将导致一直发送完成消息，直到我们系统响应不了，

悲剧就会发生（这时只能重启电脑，要不什么东西都不能用了）。

PostQueuedCompletionStatus(m\_hCompletionPort, 0, (DWORD) NULL, NULL);

// Sleep(60);

// Check if the IOWorkers are terminated..

POSITION pos = m\_IOWorkerList.GetHeadPosition ();

//这里就是问题发生的所在，因为如果工作者线程数量为0，pos将始终为NULL，bIOWorkersRunning的值也不会被改变，所以觉得有必要要先判断当前的工作者线程的数量。//PS：测试方法 调用IOCPS::Start方法时将iMaxIOWorkers参数设定为0。（又或者某种错误发生使工作者线程数减少到0呢？）

while (pos != NULL)

{……}

第2个问题：发生在IOCPS.cpp的void IOCPS::DisconnectClient(ClientContext \*pContext, BOOL bGraceful)方法上

void IOCPS::DisconnectClient(ClientContext \*pContext, BOOL bGraceful)

{

if(pContext!=NULL)

{

pContext->m\_ContextLock.Lock(); //这个地方加锁了

BOOL bDisconnect=pContext->m\_Socket!=INVALID\_SOCKET;

pContext->m\_ContextLock.Unlock(); //但是这里解锁了，这意味着如果有多个工作者线程的话，当同时处理到这里的话，就可能使bDisconnect都为TRUE

//也就是说，对于同一个ClientContext，下面的if模块可能会被执行多次

…….  
// Notify that we are going to Disconnect A client.   
NotifyDisconnectedClient(pContext); //那么对于同一个pContext断开连接的通知也可能多于一次，这肯定不是我们想要的。这就是我所说的缺陷

第3个问题：这个缺陷在在IOCPS.cpp文件的void IOCPS::OnInitialize(ClientContext \*pContext, DWORD dwIoSize,CIOCPBuffer \*pOverlapBuff)方法。 提高这个问题发生的条件是：在测试的时候，创建多个套接字（比如2000个socket）然后去连接服务器，连接上以后，客户端马上调用closesocket关闭。

void IOCPS::OnInitialize(ClientContext \*pContext, DWORD dwIoSize,CIOCPBuffer \*pOverlapBuff)

{ ……..

AZeroByteRead(pContext,pOverlapBuff); //因为是客户端连接上以后，马上调用closesocket关闭连接，AZeroByteRead调用会投递一个IOZeroByteRead完成消息，【If client connceted the server, then client call closesocket function at once. And call AZeroByteRead function will post a 'IOZeroByteRead' message.】

//那么在OnZeroByteRead方法里WSARecv的调用就可能失败，然后调用ReleaseClientContext回收pContext（因为此时pContext->m\_nNumberOfPendlingIO为0）。那么这种情况下，到这里就该结束了，不能再往下执行了。

for(int i=0;i&lt;m\_iNumberOfPendlingReads;i++)

{

//所以在调用EnterIOLoop之前，也应该先判断下pContext🡪m\_nNumberOfPendlingIO的值是不是0，如果是EnterIOLoop方法和ARead方法就不能再调用。如果这里pContext🡪;m\_nNumberOfPendlingIO已经为0了（也就是说pContext已经被回收了），ARead方法调用PostQueuedCompletionStatus投递IORead完成通知。那么OnRead调用肯定失败。然后也调用ReleaseClientContext回收pContext。这就是所说的m\_FreeContextList存在两个一摸一样的pContext的原因。

inline BOOL IOCPS::ReleaseClientContext(ClientContext \*pContext)

{

BOOL bRet=FALSE; if(pContext!=NULL) {

// We are removing this pContext from the penling IO port.

int nNumberOfPendlingIO=ExitIOLoop(pContext);

// We Should not get an EnterIOLoopHere Because the client are disconnected.

#ifdef \_DEBUG

if(nNumberOfPendlingIO<0) //请看IOCPS::OnInitialize的描述，所以nNumberOfPendling<0就可能成立

……..

if(m\_FreeContextList.GetCount()&lt;m\_iMaxNumberOfFreeContext||m\_iMaxNumberOfFreeContext==0) {

bRet=m\_FreeContextList.AddHead((void\*)pContext)!=NULL; //如果pContext已经在m\_FreeContextList里了呢？

…….}

1. the function "void IOCPS::**ShutDownIOWorkers()**"

in the inner while loop, if one thread make bIOWorkersRunning equal to TRUE, and later, the last thread make it to FALSE, and then bIOWorkersRunning is FALSE finally, so it will not enter into the outside while loop. i wonder if all IOWorkers are terminated.

作者：No where in the IOCPS class threads the bIOWorkersRunning is set to true. Yes all IOWorkers are terminated. They also terminate if the IOCPS port is closed.

其他人：My god ,I have the problem too。

1. m\_iMaxIOWorkers>1, blocks file transfer

am testing code with several configurations, transferring files many times to see if it is stable or not... When I change the m\_iMaxIOWorkers>1, for example 10, the transfer blocks in some stage of files transfer. With m\_iMaxIOWorkers=1, I could transfer thousands of documents with no problem.

作者：Sounds like you have a deadlock in your protocol that you apply on top of IOCPS.h class. I have never experienced this (not even on a dual core mashine) Do you get this when you run the demo example or your own code? Read: 7 Special considerations & rule of thumbs Be careful of the m\_iMaxIOworker. When you do a transmit file a low level kernel thread is used to send all of the data but if you occupy to many threads with asynchronously read (e.g. m\_iMaxIOworker > 10) you will get some problems.

The m\_iMaxIOworker should be the same number as number of CPU (or core). You can acctually assign the threads to different CPU or core. (not implemented in this example)

Anyone have a similar behaviour?

1. ProcessPackage函数：用WSARecv接受完数据后，在ProcessPackage函数中对数据进行相应的验证与处理
2. 首先判断是否为分批发送数据（Pending reads），如果不是，则转到2），否则转到8）
3. 取得所接受数据的总长度nUsedBuffer，如果长度不小于MINIMUMPACKAGESIZE（sizeof（UINT）），则转到3），否则转到7）
4. 如果取数据包中的前四个字节所对应的数据长度nSize==nUsedBuffer-MINIMUMPACKAGESIZE.则数据格式正确，则通过函数NotifyReceivedPackage对接受的数据进行相应的处理，退出循环。
5. 如果nUsedBuffer-MINIMUMPACKAGESIZE >nSize，说明接受了更多的数据（We have more data），调用函数SplitBuffer对buffer进行拆分成指定大小，调用NotifyReceivedPackage函数，释放缓冲区，进行循环
6. 如果nUsedBuffer-MINIMUMPACKAGESIZE<nSize && nSize<MAXIMUMPACKAGESIZE，说明数据包在本次的数据流和下次的数据流中重叠（The package is overlapped between this byte chunk stream and the next），退出循环
7. 如果nSize>MAXIMUMPACKAGESIZE，说明接受的有效数据长度大于规定的最大接受长度（可能原因：数据格式不对或者是恶意攻击），释放资源，将本次的套接字添加到禁止列表（如果在IOCPS.h中定义了define SIMPLESECURITY），断开连接。
8. nUsedBuffer>0 && nUsedBuffer<MINIMUMPACKAGESIZE, 说明数据长度太小，连数据头部都没有包含，退出循环
9. 计算数据大小nUsedBuffer，如果nUsedBuffer<MINIMUMPACKAGESIZE,说明数据长度太小容不下数据包中数据表头，进行AddAndFlush操作，否则转到9）
10. 如果nSize<=(MAXIMUMPACKAGESIZE-MINIMUMPACKAGESIZE)，说明数据格式正确，计算额外需要的空间，进行AddAndFlush操作