Winsock I/O Methods

- In addition to using **WSAAsyncSelect** model, Windows also provides two other I/O models.
- These are **Overlapped I/O** and **Completion Port I/O**.

Overlapped I/O

- The main advantage of overlapped I/O is that it allows an application to perform **asynchronous network I/O**, which results in an **enhanced data throughput**.
- This technique **avoids extra data copies** during send and receive operations by copying data directly between an application-specified buffer and the low-level network interface.
- It also allows for more **flexibility in timing relationships** between socket-based applications and the network over which data is being exchanged.
- The basic design of the overlapped model allows an application to post one or more Winsock I/O requests using an overlapped data structure. At some later time, the application will service submitted requests after they have completed.
- As an example consider a multimedia application that generates an audio stream between two hosts on a network. Generally, this sort of an application generates fixed amount of data to be sent at regular intervals.
- Networks on the other hand will provide inconsistent bandwidth capacity so they cannot always accept the stream when the application is ready to send it. This results in what is known as packet "clumping".
- Using overlapped I/O an application can create a buffer, post a routine to fill the buffer as data arrives and at the same time perform other operations while the buffer is being filled. This technique can be applied at both the send and receive ends.
- Thus, asynchronous operations using overlapped I/O smooth out the timing differences between bursty network traffic and constant-rate multimedia streams.
- To use the overlapped I/0 model on a socket, the application must first create a socket by using the **WSA_FLAG_OVERLAPPED** flag as follows:

s = WSASocket (AF_INET, SOCK-STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);

- After the socket has been successfully created, and the application has issued a bind, overlapped I/O operations can now be performed on the socket by calling one or more in a suite of Winsock functions and specifying an optional **WSAOVERLAPPED** structure.

    - **WSASend**
    - **WSASendTo**
    - **WSARecv**
    - **WSARecvFrom**
    - **WSAIoctl**
    - **AcceptEx**
    - **TransmitFile**

- Each one of the above functions is associated with sending data, receiving data, and accepting connections on a socket.

- Since this activity can potentially take a long time to complete, each function can accept a **WSAOVERLAPPED** structure as a parameter.

- When these functions are called with a **WSAOVERLAPPED** structure, they complete immediately, regardless of whether the socket is set to blocking mode.

- They rely on the **WSAOVERLAPPED** structure to manage the return of an I/O request.

- There are essentially two methods for managing the completion of an overlapped I/O request:

    - The application can wait for **event object notification**

    - Or it can process completed requests through **completion routines**.

- All the functions listed above (except ***AcceptEx***) have another parameter in common: **lpCompletionROUTINE**.

- This parameter is an optional pointer to a completion routine function that gets called when an overlapped request completes.

Event notification

- The event notification method of overlapped I/O requires associating Win32 event objects with **WSAOVERLAPPED** structures.

- When I/O calls such as ***WSASend*** *and* ***WSARecv*** are made using a **WSAOVERLAPPED** structure, they return immediately.

- Usually these I/O calls fail with the return value **SOCKET_ERROR**. The ***WSAGetLastError*** function reports a **WSA_IO_PENDING** error status.

- This error status simply means that the I/O operation is in progress. At some later time, the application will need to determine when an overlapped I/O request completes by waiting on the event object associated with the **WSAOVERLAPPED** structure.

- The **WSAOVERLAPPED** structure provides the communication medium between the initiation of an overlapped I/O request and its subsequent completion, and is defined as

```
typedef struct WSAOVERLAPPED
{
        DWORD       Internal;
        DWORD       InternalHigh;
        DWORD       Offset;
        DWORD       OffsetHigh;
        WSAEVENT    hEvent;
}WSAOVERLAPPED, FAR * LPWSAOVERLAPPED;
```

- The **Internal**, **InternalHigh**, **Offset**, and **OffsetHigh** fields are all used internally by the system and should not be manipulated or used directly by an application.

- The **hEvent** field, on the other hand, is a special field that allows an application to associate an **event object handle** with a **socket**.

- The application can use the ***WSACreateEvent*** function to create an event object handle.

- Once an event handle is created, simply assign the overlapped structure's ***hEvent*** field to the event handle and start calling a Winsock functions such as ***WSASend*** *or* ***WSARecv*** using the overlapped structure.

- When an overlapped I/O request finally completes, the application is responsible for retrieving the overlapped results.

- In the event notification method, Winsock will change the event-signaling state of an event object that is associated with a **WSAOVERLAPPED** structure from **nonsignaled** to **signaled** when an overlapped request finally completes.

- Because an event object is assigned to the **WSAOVERLAPPED** structure, we can easily determine when an overlapped I/O call completes by calling the ***WSAWaitForMultipleEvents*** function.

- ***WSAWaitForMultipleEvents*** waits a specified amount of time for one or more event objects to become signaled. It is important to remember that ***WSAWaitForMultipleEvents*** is capable of waiting on only 64 event objects at a time.

- Once we have determined which overlapped request have completed, we need to determine the success or failure of the overlapped call by calling ***WSAGetOverlappedResult:***

```
BOOL WSAGetOverlappedResult (
        SOCKET s,
        LPWSAOVERLAPPED lpOverlapped,
        LPDWORD lpcbTransfer,
        BOOL fWait,
        LPDWORD lpdwFlags
);
```

- **s** identifies the socket that was specified when the overlapped operation was started.

- **lpOverlapped** is a pointer to the **WSAOVERLAPPED** structure that was specified when the overlapped operation was started.

- **lpcbTransfer** is a pointer to a **DWORD** variable that receives the number of bytes that were actually transferred by an overlapped send or receive operation.

- **fWait** determines whether the function should wait for a pending overlapped operation to complete.

- If **fWait** is TRUE, the function does not return until the operation has been completed.

- If **fWait** *is* FALSE and the operation is still pending, ***WSAGetOverlappedResult*** returns FALSE with the error **WSA_IO_INCOMPLETE.** Since in our case we waited on a signaled event for overlapped completion, this parameter has no effect.

- **lpdwFlags** is a pointer to a **DWORD** that will receive resulting flags if the originating overlapped call was made with the ***WSARecv*** or the ***WSARecvFrom*** function.

- If the ***WSAGetOverlappedResult*** function succeeds, the return value is TRUE. This means that your overlapped operation has completed successfully and that the value pointed to by **lpcbTransfer** has been updated.

- If the return value is **FALSE**, one of the following statements is true:

- The overlapped I/O operation is still pending (as described above).

- The overlapped operation completed, but with errors.

- The overlapped operation's completion status could not be determined because of errors in one or more of the parameters supplied to ***WSAGetOverlappedResult.***

- Upon failure, the value pointed to by **lpcbTransfer** will not be updated, and the application should call the ***WSAGetLastError*** function to determine the cause of the failure.

- The code fragment provided illustrates how to design a simple server application that is capable of managing overlapped I/O on one socket, using the event notification described above.

- The application can be summarized as a set of the following programming steps:

    1. Create a socket, and begin listening for a connection on a specified port.

    2. Accept an inbound connection.

    3. Create a **WSAOVERLAPPED** structure for the accepted socket, and assign an event object handle to the structure.

       Also assign the event object handle to an event array to be used later by the ***WSAWaitForMultipleEvents*** function.

    4. Post an asynchronous ***WSARecv*** request on the socket by specifying the **WSAOVERLAPPED** structure as a parameter.

       NOTE: This function will normally fail with **SOCKET_ERROR** error status **WSA_I/O_PENDING.**

    5. Call ***WSAWaitForMultipleEvents*** using the event array, and wait for the event associated with the overlapped call to become signaled.

    6. After ***WSAWaitForMultipleEvents*** completes, reset the event object by using ***WSAResetEvent*** with the event array, and process the completed overlapped request.

    7. Determine the return status of the overlapped call by using ***WSAGetOverlappedResult.***

    8. Post another overlapped ***WSARecv*** request on the socket.

    9. Repeat steps 5-8.

- This example can easily be expanded to handle more than one socket by moving the overlapped I/O processing portion of the code to a separate thread and allowing the main application thread to service additional connection requests.

Completion Routines

- Completion routines are the other method your application can use to manage completed overlapped I/O requests.

- Completion routines are simply functions that are optionally passed to an overlapped I/O request and that the system invokes when an overlapped I/O request completes.

- Their primary role is to service a completed I/O request using the caller's thread. Additionally, applications can continue overlapped I/O processing through the completion routine.

- To use completion routines for overlapped I/O requests, the application must specify a completion routine, along with a **WSAOVERLAPPED** structure, to an I/O bound Winsock function.

- The completion routine must have the following function prototype:

```
void CALLBACK CompletionROUTINE (
    DWORD dwError,
    DWORD cbTransferred,
    LPWSAOVERLAPPED lpOverlapped,
    DWORD dwFlags
);
```

- When an overlapped I/O request completes using a completion routine, the parameters contain the following information:

    - **dwError** specifies the completion status for the overlapped operation as indicated by **lpOverlapped**.

    - **cbTransferred** parameter specifies the number of bytes that were transferred during the overlapped operation.

    - **lpOverlapped** parameter is the **WSAOVERLAPPED** structure passed into the originating I/O call.

    - **dwFlags** parameter is not used and will be set to 0.

- There is a major difference between overlapped requests submitted with a completion routine and overlapped requests submitted with an event object.

- The **WSAOVERLAPPED** structure's event field, **hEvent**, is not used, which means you cannot associate an event object with the overlapped request.

- Once you make an overlapped I/O call with a completion routine, your calling thread must eventually service the completion routine once it has completed.

- This requires you to place your calling thread in an **alertable wait state** and process the completion routine later, after the I/O operation has completed.

- The ***WSAWaitForMultipleEvents*** function can be used to put your thread in an alertable wait state.

- However, you must also have at least one event object available for the ***WSAWaitForMultipleEvents*** function.

- If the application handles only overlapped requests with completion routines, it is not likely to have any event objects around for processing.

- As an alternative, the application can use the Win32 ***SleepEx*** function to set the thread in an alertable wait state. It can also create a dummy event object that is not associated with anything.

- If the calling thread is always busy and not in an alertable wait state, no posted completion routine will ever get called.

- The ***WSAWaitForMultipleEvents*** normally waits for event objects associated with **WSAOVERLAPPED** structures.

- This function is also designed to place the thread in an alertable wait state and to process completion routines for completed overlapped I/O requests if the **fAlertable** parameter is set to **TRUE.**

- When overlapped I/O requests complete with a completion routine, the return value is **WSA_IO_COMPLETION** instead of an event object index in the event array.

- The ***SleepEx*** function provides the same behavior as ***WSAWaitForMultipleEvents*** except that it does not need any event objects.

- The ***SleepEx*** function is defined as

  DWORD SleepEx (
  DWORD dwMilliseconds,
  BOOL bAlertable
  );

  - **dwMilliseconds** parameter defines how long in milliseconds ***SleepEx*** *will* wait. If it is set to INFINITE, ***SleepEx*** waits indefinitely.

  - **bAlertable** parameter determines how a completion routine will execute.

  - If it is set to FALSE and an I/O completion callback occurs, the I/O completion function is not executed and the function does not return until the wait period specified in **dwMilliseconds** has elapsed.

  - If it is set to TRUE, the completion routine executes and *the* ***SleepEx*** function returns **WAIT_IO_COMPLETI0N**.

- The code fragment provided illustrates how to design a simple server application that is capable of managing a socket request using the completion routines described above.

- The application can be summarized as a set of the following programming steps:

    1. Create a socket and begin listening for a connection on a specified port.
    2. Accept an inbound connection.
    3. Create a **WSAOVERLAPPED** structure for the accepted socket.
    4. Post an asynchronous ***WSARecv*** request on the socket by specifying the **WSAOVERLAPPED** structure as a parameter and supplying a completion routine.
    5. Call ***WSAWaitForMultipleEvents*** with the **fAlertable** parameter set to TRUE, and wait for an overlapped request to complete.

       When an overlapped request completes, the completion routine automatically executes and ***WSAWaitForMultipleEvents*** returns **WSA_IO_COMPLETION**.

       Inside the completion routine, post another overlapped ***WSARecv*** request with a completion routine.
    6. Verify that ***WSAWaitForMultipleEvents*** returns **WSA_IO_COMPLETION**.
    7. Repeat steps 5 and 6.

Summary of I/O Models

- We have covered the various I/O models available in Winsock 2. These models allow applications to tailor Winsock I/O according to specific needs, from simple blocking I/O to high-performance I/O for the maximum throughput.

- Each model has its strengths and weaknesses. All of the I/O models require fairly complex programming when compared with developing a simple blocking-mode application with many servicing threads.

- We can follow a general set of guidelines for client and server development.

Client Design

- When implementing a client application that manages one or more sockets, **overlapped I/O** or ***WSAEventSelect*** is recommended over the other I/O models for performance reasons.

- However, if you are developing a Windows-based application that manages window messages, the ***WSAAsyncSelect*** model might be a better choice because ***WSAAsyncSelect*** lends itself to the Windows message model, and the application is already set up for handling messages.

Server Design

- When developing a server that processes several sockets at a given time, using **overlapped I/O** is recommended over the other I/O models for performance reasons.