

# Multithreading Applications in Win32

(Chapter 6. Overlapped I/O or  
Juggling Behind Your Back)

Eun-Sung , Lee  
twoss@mmlab.net

[Multimedia Lab.](#)

Dept. of Electrical and Computer Eng.  
University of Seoul  
Seoul, Korea

# Contents

---

6.0 Main Objectives

6.1 Overview

6.2 The Win32 File Calls

6.3 -Signaled File Handles

6.4 -Signaled Event Objects

6.5 -A Synchronous Procedure Calls(APCs)

6.6 Drawbacks to Overlapped I/O with Files

6.7 -I/O Completion Ports

6.8 Using Overlapped I/O with Sockets

6.9 Summary

# 6.0 Main Objectives

---

- How to use overlapped, or asynchronous, I/O.

# 6.1 Overview

---

- Waiting for I/O to complete is not a particularly good use of time for your program.
- An obvious solution to this problem is would be to use another thread to do the I/O
  - (problem : synchronization , handling errors , put up dialogs ...)
- Overlapped I/O in Win32
  - You can ask the operating system to transfer data for you and to notify you when it is finished

# 6.1 Overview

---

- Non-overlapped VS overlapped
  - ( I/O Devices are slow ! )
  - Non-overlapped : it is not a particularly good use of time
    - An awfully long time is needed to finish for an application
  - Overlapped : you can ask the operating system to transfer data for you and to notify when it is finished
    - This arrangement leaves your application free to continue processing while the I/O is being performed

Overlapped I/O(Windows NT only)

- WIN95 does not support overlapped I/O completely  
= asynchronous, non-blocking I/O (other OS)

# 6.1 Overview

---

- Signaled file handles
- Signaled event objects
- Asynchronous Procedure Calls(APCs)
- I/O completion ports
  - Important !! Because they are the only mechanism that is suited for high volume-servers that must maintain many simultaneous connections

## 6.2 The Win32 File Calls

---

- Three basic calls for performing I/O in Win32
  - CreateFile()
  - ReadFile()
  - WriteFile()
    - There is no way for closing a file ??
    - You can use CloseHandle() to do this
- CreateFile() can be used to open a wide variety of resources
  - Files(hard disk , floppy disk , CD-ROM , and others
  - Serial and parallel ports
  - Named pipes
  - Console (CH.8)

## 6.2 The Win32 File Calls

---

### 1. CreateFile()

Prototype :

```
HANDLE CreateFile(
    LPCTSTR lpFileName,
    DWORD   dwDesiredAccess,
    DWORD   dwShareMode,
    LPSECURITY_ATTRIBUTES //lpSecurityAttributes,
    DWORD   dwCreationDistribution, //how to create
    DWORD   dwFlagsAndAttributes, //file Attributes
    HANDLE  hTemplateFile
);
```

## 6.2 The Win32 File Calls

---

### 1. CreateFile()

- dwFlagAndAttributes (6<sup>th</sup> parameter)
  - : the Key to using overlapped I/O.
  - : `FILE_FLAG_OVERLAPPED`
  - **I f!** this flag is specified, then every operation on the file handle must be overlapped.

An unusual feature of overlapped I/O is that it can read and/or write from multiple parts of the file at the same time

- There is no way to use overlapped I/O with the calls in `stdio.h`. therefore there is no easy way of doing overlapped text-based I/O

## 6.2 The Win32 File Calls

---

```
2. BOOL ReadFile(
    HANDLE hFile,
    LPVOID lpBuffer,
    DWORD nNumberOfBytesToRead,
    LPDWORD lpNumberOfBytesRead,
    LPOVERLAPPED lpOverlapped //ptr to overlapped info
);
```

```
3. BOOL WriteFile(
    HANDLE hFile,
    LPCVOID lpBuffer,
    DWORD nNumberOfBytesToWrite,
    LPDWORD lpNumberOfBytesWritten,
    LPOVERLAPPED lpOverlapped //ptr to overlapped info
);
```

## 6.2 The Win32 File Calls

---

- Similar to `fread()` and `fwrite()` except for the last parameter
- You must supply a pointer to a valid `OVERLAPPED` structure of `CreateFile()` was called with `FILE_FLAG_OVERLAPPED`
- **The `OVERLAPPED` Structure**
  - Acts as a key that uniquely identifies each overlapped operation currently in process.
  - ~provides a shared area between you and the system where parameters can be passed in each direction.

## 6.2 The Win32 File Calls

```
typedef struct _OVERLAPPED{  
    DWORD   Internal;  
    DWORD   InternalHigh;  
    DWORD   Offset;  
    DWORD   OffsetHigh;  
    HANDLE  hEvent;  
} OVERLAPPED;
```

Member	Description
Internal	Reserved for operating system use. This member, which specifies a system-dependent status, is valid when the <u>GetOverlappedResult</u> function returns without setting the extended error information to ERROR_IO_PENDING.

## 6.2 The Win32 File Call

Member	Description
InterHigh	Reserved for operating system use. This member, which specifies the length of the data transferred, is valid when the <b>GetOverlappedResult</b> function returns TRUE.
Offset	Specifies a file position at which to start the transfer. The file position is a byte offset from the start of the file. The calling process sets this member before calling the <a href="#">ReadFile</a> or <a href="#">WriteFile</a> function. This member is ignored when reading from or writing to named pipes and communications devices and should be zero.
OffsetHigh	Specifies the high word of the byte offset at which to start the transfer. This member is ignored when reading from or writing to named pipes and communications devices and should be zero.

## 6.2 The Win32 File Call

Member	Description
hEvent	<p>Handle to an event set to the signaled state when the operation has been completed. The calling process must set this member either to zero or a valid event handle before calling any overlapped functions. To create an event object, use the <a href="#">CreateEvent</a> function. Functions such as <b>WriteFile</b> set the event to the nonsignaled state before they begin an I/O operation.</p>

- It is important to put the structure in a safe place
- Typically a local variable is not a safe place because it will go out of scope too soon.
- It is usually safest to allocate the OVERLAPPED structure on the heap

## 6.3 Signaled File Handles

---

- The simplest type of overlapped I/O operation
- The file handle, as a kernel object, will be signaled when the operation is complete.

### Step:

1. Opening the file with `FILE_FLAG_OVERLAPPED`
2. Set up an `OVERLAPPED` structure
3. Call `ReadFile()` and pass the address of the `OVERLAPPED` structure
4. `GetOverlappedResult()` to find out what happened,  
`GetLastError()`.

## 6.3 Signaled File Handles

---

Prototype :

```
BOOL GetOverlappedResult(  
    HANDLE hFile,  
    LPOVERLAPPED lpOverlapped,  
    LPDWORD lpNumberOfBytesTransferred,  
    BOOL bWait //True -> non-Overlapped  
)
```

Return Value

True : overlapped operation succeed

False : failed. -> GetLastError()

Let's see the Listing 6-1. (p.123)

## 6.3 Signaled File Handles

---

- **Notice**

- Asking for an operation to be overlapped does not mean that it will be overlapped.
- If you ask for an operation to be overlapped, and the OS queues the request for execution, then `ReadFile()` and `WriteFile()` return `FALSE` to indicate failure.

: so `GetLastError()`

`ERROR_IO_PENDING` : simply means that the overlapped I/O is queued and waiting to happen.

`ERROR_HANDLE_EOF` : really an error

## 6.4 Signaled Event Objects

---

- A limitation to using file handles as the signaling mechanism is that you can't tell which operation has completed
  - > you cannot tell which operation has completed (several operations, same file handle)
- Solution :
  1. GetOverlappedResult()
  2. OVERLAPPED structure's hEvent
    - Ex) set to NULL: Using a signaled file handle  
hEvent : kernel will automatically signal the event object when the overlapped I/O operation completed

## 6.4 Signaled Event Objects

---

- Example)

Let's see Listing 6-2. (p.126)

- `ERROR_INVALID_USER_BUFFER`  
: there are insufficient resources.  
-> The number of works Controlled by OS for performance

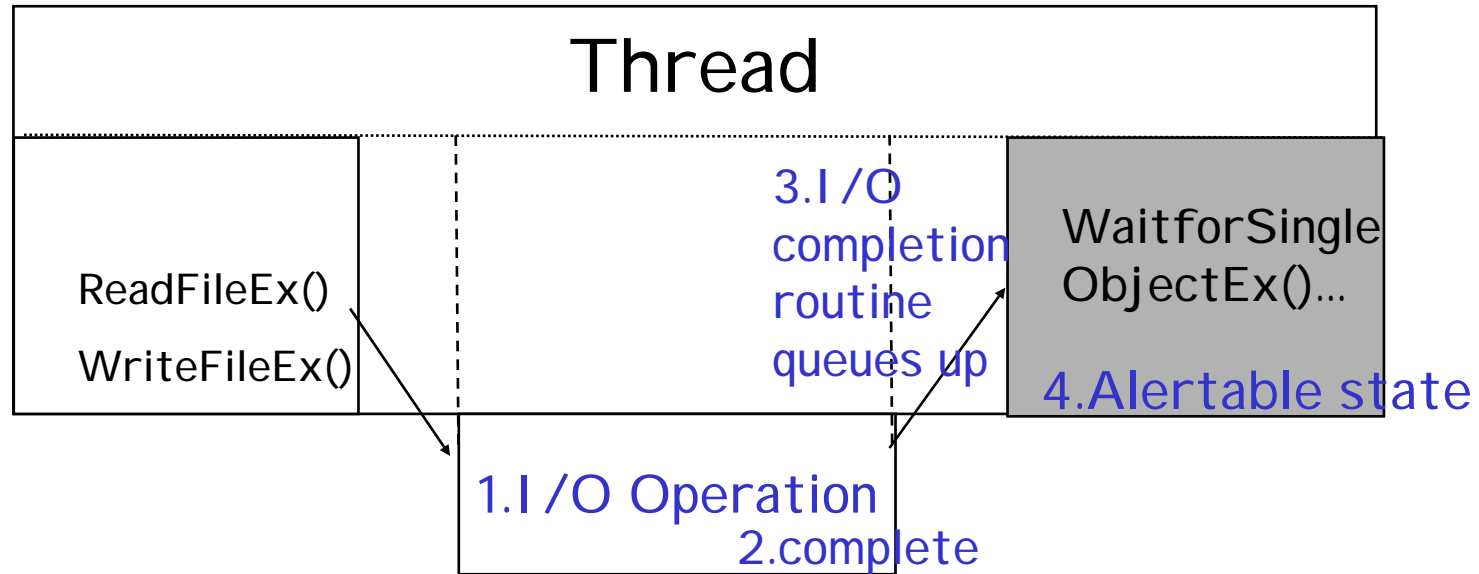
# 6.5 Asynchronous Procedure Calls(APCs)

---

- Problems :
  - WaitForMultipleObject() ->  
MAXIMUM\_WAIT\_OBJECTS = 64
  - Constantly trying to figure out how to react base on which handle was signaled. ( WaitForMultipleObjcet() )
- Solution : **APC**
  - Using the "Ex" versions of ReadFile(), WriteFile()  
( Read FileEx(), Write FileEx()... )
  - Take an additional parameter that specifies a callback routine(**I/O completion routine**)  
: the system should call when an overlapped I/O operation finishes.

# 6.5 Asynchronous Procedure Calls(APCs)

- Alertable state :



- APCs can only be called when the application is in an alertable state.
  - `SleepEx()`, `WaitForSingleObjectEx()`, `WaitForMultipleObjectEx()`, `MsgWaitForMultipleObjectEx()`, `SignalObjectAndWait()`

## 6.5 Asynchronous Procedure Calls(APCs)

---

- VOID WINAPI FileIoCompletionRoutine(  
    DWORD dwErrorCode,  
    DWORD dwNumberOfBytesTransferred,  
    LPOVERLAPPED lpOverlapped  
)

## 6.5 Asynchronous Procedure Calls(APCs)

---

- Step:
  1. CreateEvent()
  2. CreateFile()
  3. ReadFileEx()
  4. WaitForSingleObjectEx()
  5. File IOCompletionRoutine()
- Let's See Example (Listing 6-3)
- ?? How do I use a not static C++ member function as an I/O completion routine?
  - (This is similar to the problem of starting a thread in a C++ member function)

## 6.6 Drawbacks of Overlapped I/O with Files

---

- Windows NT decides whether or not to queue an operation based on the size of the request. (64k)

(ex) transfer rate 5MB/sec

8KB -> block : 10ms ( :seek the head)

-> transfer : 1.8ms

Nowadays : very different (should be 311k , 360k)

- In a Web server, using overlapped I/O would decrease overall performance.

- -> bypass the virtual memory manager

- -> go directly to the file system

(CreateFile() with FILE\_FLAG\_NO\_BUFFERING)

## 6.6 Drawbacks of Overlapped I/O with Files

### IBM Deskstar 75GXP and Deskstar 40GV at a glance

Model	Deskstar 75GXP	Deskstar 40GV
	DTLA-307075/307060/ 307045/307030/ 307020/307015	DTLA-305040/305030/ 305020

### Configuration

Interface	ATA	ATA
Capacity (GB)	75/60/45/30/20/15	40/30/20
Sector size (bytes)	512	512
Recording zone	15	15
User cylinders (physical)	27,724	34,326
Data heads (physical)	10/8/6/4/3/2	4/3/2
Data disks	5/4/3/2/2/1	2/2/1
Max. areal density (Gbits/sq. inch)	11.0	14.5
Max. recording density (BPI)	391,000	415,000
Track density (TPI)	28,350	35,000

### Performance

Data buffer	2 MB <sup>2</sup>	512 KB <sup>2</sup>
Rotational speed (RPM)	7200	5400
Latency (average ms)	4.17	5.56
Media transfer rate (max Mbits/sec)	444	372
Interface transfer rate (max MB/sec)	100	100
Sustained data rate (MB/sec)	37	32
Seek time <sup>3</sup> (read typical)		
Average (ms)	8.5	9.5
Track-to-track (ms)	1.2	1.6
Full-track (ms)	15.0	16.0

# 6.7 I/O Completion Ports

---

- Problems of APCs with the overlapped I/O
  - Several I/O APIs doesn't supports APCs.
  - Only the thread that started the overlapped request can service the callback.
- Solution : In NT 3.5~, **I/O Completion ports**
  - Completion ports solve all of the problems we have seen so far
  - There is no limit to the number of handles.
  - It allows one thread to queue a request and another thread to service it.
  - implicitly support scalable architectures.

# 6.7 I/O Completion Ports

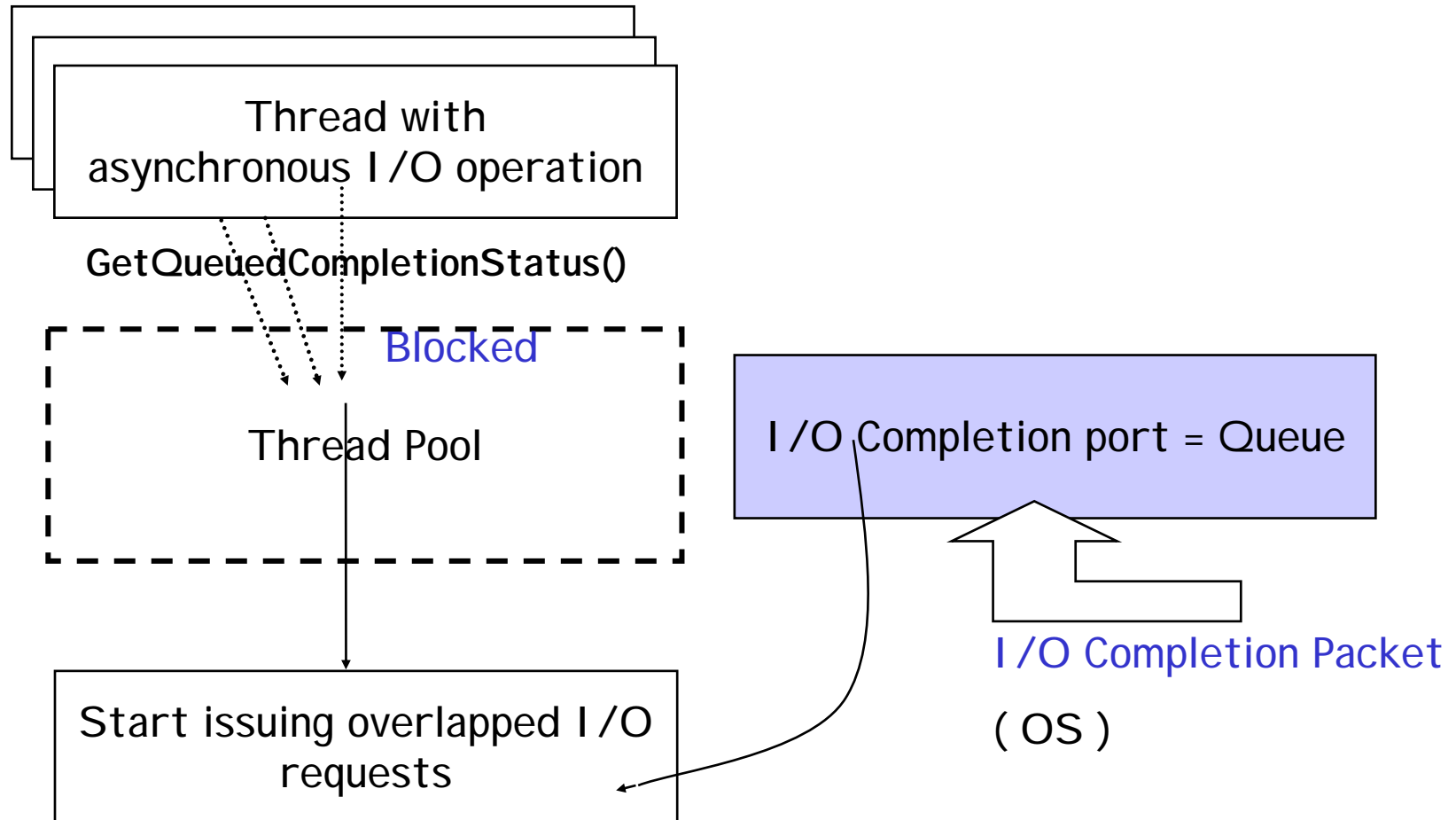
---

- **Server Threading Models**
  - There are three basic ways of deciding how many threads to use in a server
    - One Single Thread
    - One Single per Client
    - One Active Thread per Processor

=> Use I/O Completion Ports !
  - **I/O Completion Ports** is a very special kind of **kernel object** that coordinates how a pool of threads services overlapped request, even across multiple processors.
- **What completion port do??**
  - The way an I/O completion port works is radically different from any of the others.

# 6.7 I/O Completion Ports

- I/O Completion Port ?



## 6.7 I/O Completion Ports

---

- Overview of Operation

1. Create the I/O Completion port.
2. Associate file handle with it.
3. Create a pool of threads.
4. Make each thread wait on the completion port.
5. Start issuing overlapped I/O requests with the file handles.

# 6.7 I/O Completion Ports

---

## 1. Creating an I/O Completion Port

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

FileHandle : INVALID\_HANDLE\_VALUE

ExistingCompletionPort : NULL

Return value : succeed -> return a handle to the I/O completion port

fail -> GetLastError()

## 2. Associating File Handles

Calling `CreateIoCompletionPort()` again once for each new file handle to be associated.

`ExistingCompletionPort` : the handle returned by first calls

## 3. Creating a Pool of Threads

---

Thread Currently running

- + Thread blocked (on disk I/O or a Wait...() call)
  - + Thread waiting on the completion port
- 

= Number of Threads in pool

---

- Number Threads to create =  $2 * \text{processor} + 2$

(chap 8, chap 10)

### 4. Issuing overlapped I/O request

- ConnectNamePipe()
- DeviceIoControl()
- LockFileEx()
- ReadFile()
- TransactNamePipe()
- WaitCommEvent()
- WriteFile()

There is no need for that thread to ever call  
`WaitForMultipleObject()`

## 6.7 I/O Completion Ports

---

- Preventing Completion Packets
  - You can read or write to a file commonly but you do not want the I/O completion port to be notified when the operation finishes.
  - Using the old signaled event object mechanism.

```
OVERLAPPED Overlap;
```

```
HANDLE hFile;
```

```
Char      buffer;
```

```
DWORD    dwBytesWritten;
```

```
Memset(&overlap, 0, sizeof(OVERLAPPED));
```

```
overlap.hEvent = CreateEvent(NULL, TRUE, FALSE,  
    NULL);
```

```
overlap.hEvent = (HANDLE)((DWORD)hEvent | 0x1);
```

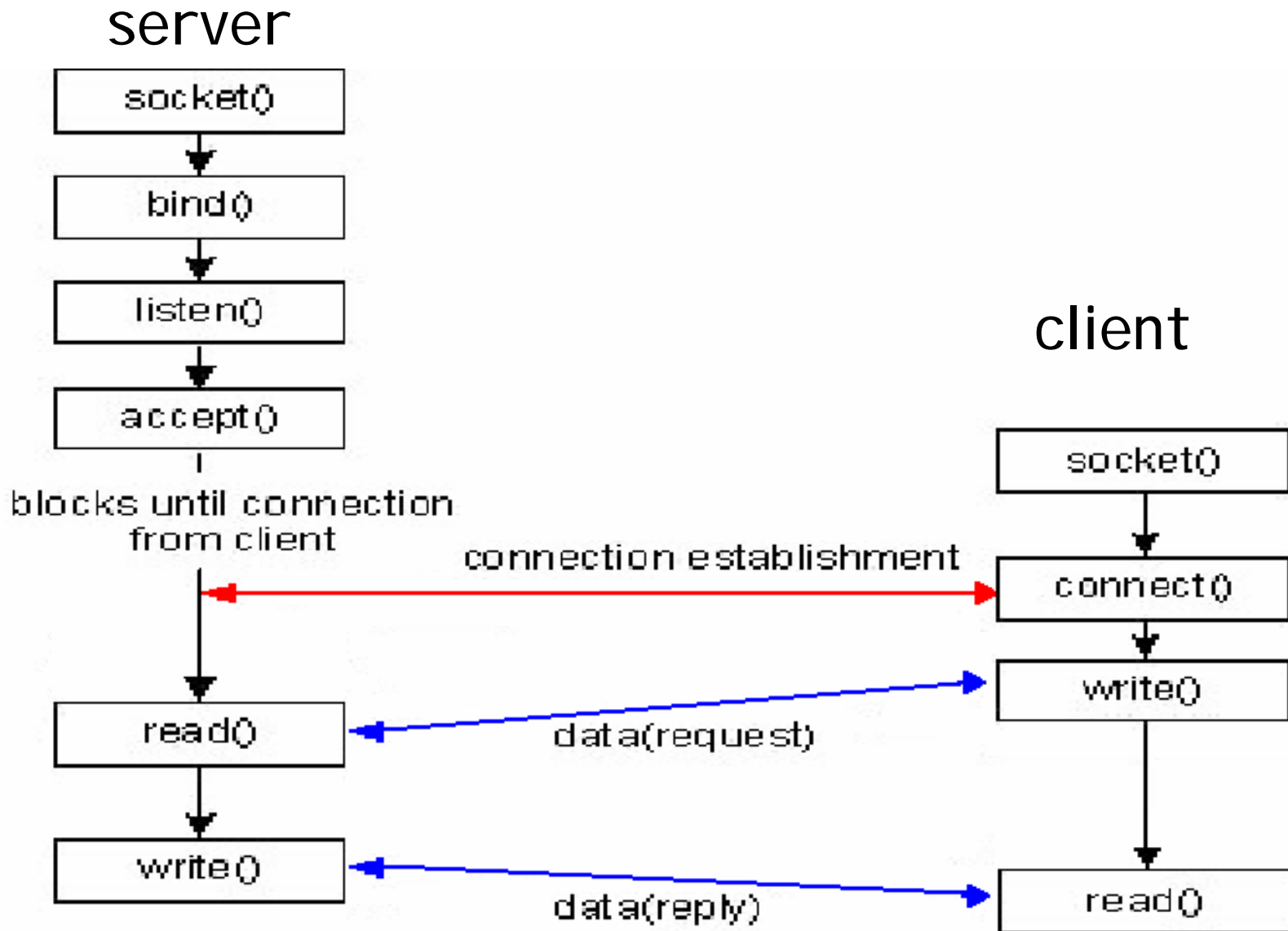
```
WriteFile(hFile, buffer, 128, dwBytesWritten, &overlap);
```

## 6.8 Using Overlapped I/O with Sockets

---

- The sample ECHO on the CD-ROM demonstrates using an I/O completion port to do the equivalent of standard TCP port 7, which echoes back every thing that is written to it.
- ECHOSRV is a server that listens on arbitrary TCP port
- ECHOCLI is the client that takes whatever you type, send ot to the server, and then print out the response
- -Completion ports are the only way to obtain high throughput on a Windows NT machine talking to the network.
  - Windows NT supports the *select()* call that is the standard way of supporting multiple simultaneous connections
  - But it has a problem with high performance.

# 6.8 Using Overlapped I/O with Sockets



## 6.8 Using Overlapped I/O with Sockets

---

- Example (Listing 6-4)

## 6.9 Summary

---

- This chapter has provided a whirlwind tour of overlapped I/O , a technique for doing I/O asynchronously that often avoids the need for using multiple threads .
- Overlapped I/O can be done using signaled file handles , signaled event objects , asynchronous procedure calls (APCs) , and I/O completion ports.
- I/O completion ports are very important because they are the preferred mechanism for creating high-performance servers that are easily scalable.