Unit 9: Windows 2000 Networking

9.3. Microsoft-specific extensions to Sockets; NetBIOS (Wnet) API

Microsoft-specific Extensions to Berkeley Sockets

- Tailored to the message-passing environment of windows
- WSA – *Windows Sockets Asynchronous* prefix
- Roots in Windows 3.1
  - Windows Sockets Committee

  - # include <winsock.h>
  - -link wsock32.dll

Request event notification for a socket

```
int PASCAL FAR WSAAsyncSelect (
            SOCKET s, HWND hWnd,
            unsigned int wMsg, long lEvent );
```

- Request a message to the window hWnd whenever any of the network events specified by the lEvent occurs.
  - Message which should be sent is specified by the wMsg parameter.
  - The socket for which notification is required is identified by s

Value	**Meaning**
FD_READ	Want to receive notification of readiness for reading
FD_WRITE	Want to receive notification of readiness for writing
FD_OOB	Want to receive notification of the arrival of out-of-band data
FD_ACCEPT	Want to receive notification of incoming connections
FD_CONNECT	Want to receive notification of completed connection
FD_CLOSE	Want to receive notification of socket closure

WSAAsyncSelect (contd.)

```
LRESULT WINAPI WndProc( HWND hWnd,
          UINT msg, WPARAM wParam, LPARAM lParam);
```

```
switch( msg ) {
          case WM_PAINT: ...
          case WM_DESTROY: ...
          case FD_ACCEPT: ...
          default: return( DefWindowProc( hWnd, msg, wParam, lParam ));
}
```

- Every window must have a window procedure
- Arguments to window procedure for notification window:
  - wParam contains socket number
  - lParam contains event code and any error that may have occured
- Event status:

  WORD wError = WSADETSELECTERROR(lParam); (wError != 0 ?)

WSAAsynchSelect (contd.)

- Report the event:

  ```
  WORD wEvent = WSAGETSELECTEVENT( lParam );
  ```

- Enabling functions reactivate WSAAsyncSelect:

  For FD_READ, FD_OOB events:

  - ReadFile(), read(), recv(), recvfrom() are enabling functions

  For WD_WRITE events:

  - WriteFile(), write(), send(), sendto() are enabling functions

- Request notification of different events:
  - Call WSAAsyncSelect() again

WSAAsyncSelect (contd.)

- Issuing a WSAAsyncSelect() for a socket cancels any previous WSAAsyncSelect() for the same socket.
  - For example, to receive notification for both reading and writing, the application must call WSAAsyncSelect() with both FD_READ and FD_WRITE, as follows:

    ```
    rc = WSAAsyncSelect(s, hWnd, wMsg, FD_READ | FD_WRITE);
    ```
- It is not possible to specify different messages for different events.
  - The following code will not work; the second call will cancel the effects of the first, and only FD_WRITE events will be reported with message wMsg2:

    ```
    rc = WSAAsyncSelect(s, hWnd, wMsg1, FD_READ);
    rc = WSAAsyncSelect(s, hWnd, wMsg2, FD_WRITE);
    ```
- To cancel all notification - i.e., to indicate that the Windows Sockets implementation should send no further messages related to network events on the socket - lEvent should be set to zero.

Use of WSAAsyncSelect - Server Side

1. Create a socket and bind your address to it
2. Call WSAAsyncSelect():
   - Request FD_ACCEPT notification
3. Call listen() – returns immediately
4. When connection request comes in:
   - Notification window receives FD_ACCEPT notification
   - Respond by calling accept()
5. Call WSAAsyncSelect():
   - Request FD_READ | FD_OOB | FD_CLOSE notifications for socket returned by accept()
6. Receiving FD_READ, FD_OOB notifications:
   - Call ReadFile(), read(), recv(), recvfrom() to retrieve the data
7. Respond to FD_CLOSE notification by calling closesocket()

Use of WSAAsyncSelect() - Client Side

1. Create a socket
2. Call WSAAsyncSelect():
   - Request FD_CONNECT notification
3. Call connect() – returns immediately
4. When FD_CONNECT notification comes in:
   - Request FD_READ | FD_OOB | FD_CLOSE notification on socket (reported in wParam)
5. When data from the server arrives:
   - Notification window receives FD_READ or FD_OOB events
   - Respond by calling ReadFile(), read(), recv(), or recvfrom()
   - Client should be prepared for FD_CLOSE notification

Get host information corresponding to an address - asynchronous version

```
HANDLE PASCAL FAR WSAAsyncGetHostByAddr (
        HWND hWnd, unsigned int wMsg,
        const char FAR * addr, int len, int type,
        char FAR * buf, int buflen );
```

asynchronous version of gethostbyaddr()

- hWnd:
  - The handle of the window which should receive a message when the asynchronous request completes.
- wMsg:
  - The message to be received when the asynchronous request completes.
- addr:
  - A pointer to the network address for the host. Host addresses are stored in network byte order.
- len:
  - The length of the address, which must be 4 for PF_INET.
- type:
  - The type of the address, which must be PF_INET.
- buf:
  - A pointer to the data area to receive the hostent data. Note that this must be larger than the size of a hostent structure. It is recommended that you supply a buffer of MAXGETHOSTSTRUCT bytes.
- buflen:
  - The size of data area buf above.

WSAAsyncGetHostByAddr (contd.)

- When the asynchronous operation is complete the application's window hWnd receives message wMsg.
- The wParam argument contains the asynchronous task handle as returned by the original function call.
  - The high 16 bits of lParam contain any error code.
  - The error code may be any error as defined in winsock.h.
  - An error code of zero indicates successful completion of the asynchronous operation.
- On successful completion, the buffer supplied to the original function call contains a hostent structure.
  - To access the elements of this structure, the original buffer address should be cast to a hostent structure pointer and accessed as appropriate.

Get host information corresponding to a hostname - asynchronous version

```
HANDLE PASCAL FAR WSAAsyncGetHostByName (
            HWND hWnd, unsigned int wMsg,
            const char FAR * name,
            char FAR * buf, int buflen );
```

- hWnd:
  - The handle of the window which should receive a message when the asynchronous request completes.
- wMsg:
  - The message to be received when the asynchronous request completes.
- Name:
  - A pointer to the name of the host.
- Buf:
  - A pointer to the data area to receive the hostent data. It is recommended that you supply a buffer of MAXGETHOSTSTRUCT bytes.
- Buflen:
  - The size of data area buf above.

asynchronous version of gethostbyname()

Get protocol information corresponding to a protocol name - asynchronous version

```
HANDLE PASCAL FAR WSAAsyncGetProtoByName (
          HWND hWnd, unsigned int wMsg,
          const char FAR * name, char FAR * buf, int buflen );
```

- hWnd
  - The handle of the window which should receive a message when the asynchronous request completes.
- wMsg
  - The message to be received when the asynchronous request completes.
- name
  - A pointer to the protocol name to be resolved.
- buf
  - A pointer to the data area to receive the protoent data. (supply a buffer of MAXGETHOSTSTRUCT bytes)
- buflen
  - The size of data area buf above.

asynchronous version of getprotobyname()

Get protocol information corresponding to a protocol number - asynchronous version

```
HANDLE PASCAL FAR WSAAsyncGetProtoByNumber (
            HWND hWnd, unsigned int wMsg,
            int number, char FAR * buf, int buflen );
```

- hWnd
  - The handle of the window which should receive a message when the asynchronous request completes.
- wMsg
  - The message to be received when the asynchronous request completes.
- number
  - The protocol number to be resolved, in host byte order.
- buf
  - A pointer to the data area to receive the protoent data (supply a buffer of MAXGETHOSTSTRUCT bytes)
- buflen
  - The size of data area buf above.

asynchronous version of getprotobynumber()

Additional Asynchronous Socket Routines

- WSAAsyncGetServByName()
- WSAAsyncGetServByPort()
- WSACancelAsyncRequest()
- WSACancelBlockingCall()
- WSACleanup()
- WSAGetLastError()
- WSAIsBlocking()
- WSASetBlockingHook(), WSAUnhookBlockingHook()
- WSASetLastError()
- WSAStartup()

WSASetBlockingHook

- Application invokes a blocking Sockets operation:
  - the Windows Sockets implementation initiates the operation and then enters a loop which is similar to the following pseudocode:

```
for(;;) {
    /* flush messages for good user response */
    while(BlockingHook()) ;
        /* check for WSACancelBlockingCall() */
    if(operation_cancelled()) break;
        /* check to see if operation complete
    if(operation_complete()) break;
        /* normal completion */
}
```

support those applications which require more complex message processing – MDI (multiple document interface) model

WNet API

- Connection Functions
  - WNetAddConnection
  - WNetAddConnection2
  - WNetAddConnection3
  - WNetCancelConnection
  - WNetCancelConnection2
  - WNetConnectionDialog
  - WNetConnectionDialog1
  - WNetDisconnectDialog
  - WNetDisconnectDialog1
  - WNetGetConnection
  - WNetGetUniversalName
- Enumeration Functions
  - WNetCloseEnum
  - WNetEnumResource
  - WNetOpenEnum
- Information Functions
  - WNetGetNetworkInformation
  - WNetGetLastError
  - WNetGetProviderName
  - WNetGetResourceInformation
  - WNetGetResourceParent
- User Functions
  - WNetGetUser

WNetAddConnection

```
DWORD WNetAddConnection(
        LPTSTR lpRemoteName, // pointer to network device name
        LPTSTR lpPassword, // pointer to password
        LPTSTR lpLocalName // pointer to local device name );
```

- connect a local device to a network resource
- successful connection is persistent
  - system automatically restores the connection during subsequent logon operations
  - lpRemoteName
    - Points to a null-terminated string that specifies the network resource to connect to.
  - lpPassword
    - Points to a null-terminated string that specifies the password to be used to make a connection. This parameter is usually the password associated with the current user.
    - NULL: the default password is used. If the string is empty, no password is used.
  - lpLocalName
    - Points to a null-terminated string that specifies the name of a local device to be redirected, such as F: or LPT1. The case of the characters in the string is not important.

WNetGetConnection

- retrieves the name of the network resource associated with a local device.

```
DWORD WNetGetConnection(
        LPCTSTR lpLocalName, // pointer to local name
        LPTSTR lpRemoteName, // pointer to buffer for remote name
        LPDWORD lpnLength // pointer to buffer size, in characters );
```

- *lpLocalName*
  - Points to a null-terminated string that specifies the name of the local device to get the network name for.
- *lpRemoteName*
  - Points to a buffer that receives the null-terminated remote name
- *lpnLength*
  - Points to a variable that specifies the size´of the buffer.