Asynchronous Notification – The WSAAsyncSelect Model

- Winsock provides an asynchronous I/O model that allows an application to receive Windows message-based notification of network events on a socket.

- This is accomplished with the **WSAAsyncSelect** function which is used to set a socket to nonblocking or asynchronous mode:

```
int WSAAsyncSelect (
    SOCKET s,
    HWND hWnd,
    u_int wMsg,
     long lEvent
);
```

  - **s** is the socket descriptor for which event notification is enabled.

  - The **hWnd** parameter is the window handle of the window to which a message should be sent when the socket state changes.
  - The **wMsg** parameter specifies the message type for the notification message(s).
  - The **lEvent** parameter is a mask identifying the events for which the application wants notification.

- To use this function an application must first create a window using the **CreateWindow** function and a default **WinProc** function to receive and process messages as usual.

- The last argument (**lEvent**) represents a bitmask that specifies individual or a combination of network events that are of interest to the application.

Event Type	Description
FD_READ	Application wants to receive notification of readiness for reading
FD_WRITE	Application wants to receive notification of readiness for writing
FD_OOB	Application wants to receive notification of the arrival of Out-Of-Band data
FD_ACCEPT	Application wants to receive notification of incoming connections
FD_CONNECT	Application wants to receive notification of a completed connection or multi-point join operation
FD_CLOSE	Application wants to receive notification of socket closure
FD_QOS	Application want to receive notification of socket Quality of Service (QOS) changes
FD_GROUP_QOS	Application wants to receive notification of socket group Quality of Service (QOS) changes (reserved for future use with socket groups)
FD_ROUTING_INTERFACE_CHANGE	Application wants to receive notification of routing interface changes for the specified destination(s)
FD_ADDRESS_LIST_CHANGE	Application wants to receive notification of local address list changes for the socket's protocol family

- The event mask may consist of any or-ed combination of events such as:

```
// request notification of incoming connections
iRc = WSAAsyncSelect (sd_listen, hWnd, WM_WINSOCK,
        FD_ACCEPT|FD_READ|FD_CLOSE);
if (iRc== SOCKET_ERROR)
{
        ErrorOutput (hWnd, szOutMsg2, IDS_WSASELERR, WSAGetLastError());
        return(0);
}
```

- For example, if you had a listening socket (**sd_listen**), you could issue the following command to receive a notification message when a connection is ready to be accepted:

```
iRc = WSAAsyncSelect (sd_listen, hWnd, WM_ASYNCSOCK,  FD_ACCEPT);
```

- Here the window specified by the handle **hWnd** will receive a message of type **WM_ASYNCSOCK** when there is a connection request ready to be accepted.

- The parameter values for the event notification messages triggered by **WSAAsyncSelect()** will be as follows:
  - **wParam:**

    socket
  - **WSAGETSELECTERROR(lParam)** or **HIWORD (lParam)**:

    0 if successful, else error code value.
  - **WSAGETSELECTEVENT(lParam) or LOWORD (lParam)**

    event, i.e., FD_READ, FD_CLOSE, etc.
- Note that it is recommended that you use the macros **WSAGETSELECTERROR** and **WSAGETSELECTEVENT** defined in **winsock.h** to extract the event and error values from **lParam**.
- So for the following command:

  iRc = WSAAsyncSelect (sd_listen, hWnd, WM_WSASYNC, FD_ACCEPT);

  the arriving **WM_WSASYNC** message will have the following parameters:
  - **wParam:**

    sd_listen
  - **WSAGETSELECTERROR(lParam)**:

    0 if successful, else error code value.
  - **WSAGETSELECTEVENT(lParam):**

    FD_ACCEPT

- The code fragment below illustrates how to process the notification message for the **FD_ACCEPT** event:

```
switch (message)
{
    ……..
    ……..

    case WM_WSASYNC:
        switch (WSAGETSELECTEVENT(lParam))
        {
            case FD_ACCEPT:
            // check if socket already closed, i.e. queued message is no
            // longer relevant and should be ignored.
            if (sd_listen == INVALID_SOCKET)
                return (0);

            // a connection request has arrived from a client.
            sd_acc = accept (wParam, NULL,NULL);
            if (sd_acc == INVALID_SOCKET) {
            ErrorOutput(hWnd,szOutMsg2,IDS_ACCERR,WSAGetLastError());
            return(0);
        }
```

- In a **Client/Server** application, using the **connect()** command on the client side is different from using the **accept()** command (server side) with respect to event completion in that the **connect()** command must be issued to initiate the connection processing.

- Then after messages are successfully exchanged across the network, the **connect()** can complete.

- The following sequence of commands and message processing needs to occur:

1. Issue **WSAAsyncSelect()** to set the socket to nonblocking and request notification for **FD_CONNECT**.

2. Issue **connect()** to initiate the connection. If the connection is successfully initiated, the **connect()** will still fail, i.e., return **SOCKET_ERROR**.

   The error code value will be set to **WSAEWOULDBLOCK** indicating the connection is in progress.

3. Process the notification message for the **FD_CONNECT** event.

   You **must** check for errors! There are several possible errors including:

   WSAECONNREFUSED: the remote side does not have an outstanding listen.
   WSAETIMEDOUT: the connection request message could not find the destination address.

- The following code fragments illustrate the recommended processing on the client side:

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT Message,
                          WPARAM wParam, LPARAM lParam)
{
        …….
        …….

    switch (message)
    {
        case WM_COMMAND:
                hMenu = GetMenu(hWnd);

                switch (wParam)
                {
                        case IDM_CONNECT:

                                // create client socket
                                sd_cl = socket (PF_INET, SOCK_STREAM, 0);
                                if(sd_cl == INVALID_SOCKET)
                                {
                                        // error occurred on socket call
                                        ErrorOutput(hWnd,szOutMsg2,
                                                        IDS_SOCKERR,WSAGetLastError());
                                        return(0);
                                }
                                ……..
                                ……..
                                ……..

                                // request connection
                                iRc=connect (sd_cl,(struct sockaddr far *)
                                        &addr_server,sizeof(addr_server));
                                if (iRc== SOCKET_ERROR)
                                {
                                        // error occurred on connect call. However
                                            expecting
                                        // to get WSAEWOULDBLOCK error which means
                                            connection
                                        // in progress.
                                        if( (iRc = WSAGetLastError()) !=
                                        WSAEWOULDBLOCK)
                                        {
                                                ErrorOutput(hWnd,szOutMsg2,
                                                        IDS_CONNERR, iRc);
                                                return(0);
                                        }
                                }

                                // now wait for the FD_CONNECT message to complete
                                the   connection
                                return(0);
```

- The processing of the **FD_CONNECT** (within **WM_WSAASYNC**) message will be as follows:

```
case WM_WSAASYNC:

        switch (WSAGETSELECTEVENT(lParam))
        {

                case FD_CONNECT:
                // connection has completed. Check for errors.
                if ( (iRc=WSAGETSELECTERROR(lParam)) == 0)
                {
                        // successful connection
                        …….
                        …….
                        return(0);
                 }
                else
                {
                        // Report error on the window
                        return(0);
                }
```

- Another useful asynchronous operation in a Client/Server environment is the read operation on a socket.

- The **FD_READ** event is enabled causing a message to be posted when data is available for reading in a socket.

- Stated another way, if you specify the **FD_READ** event in the **WSAAsyncSelect()** function, you will receive a message for the **FD_READ** event when there is data in the socket.

- If more data arrives and you have NOT done a **recv()** (or another **WSAAsyncSelect()**), you will NOT receive another message.

- If you do a **recv()** which only gets a portion of the available data, the next **FD_READ** event message will be posted immediately.

- Recall that a socket **recv()** call can receive any amount of data up to the length specified in the **len** parameter.

- In order to receive the full receiver buffer length, multiple **recv()**s may be required.

- The following code fragment illustrates the processing of an asynchronous read on a socket:

```
#define RECVSIZE 8192
char sRcvBuff[RECVSIZE];
int iLen, iRc;
static int iRcvStart = 0;
static int iRcvLen = RECVSIZE;
int iRLen;

//Assume the FD_READ message below was triggered by
// WSAAsyncSelect (sd1, hWnd, WM_WSAASYNC,  FD_READ);

case WM_WSAASYNC:
        switch (WSAGETSELECTEVENT(lParam))
        {
                case FD_READ:
                // check if socket already closed, i.e., message no
                // longer relevant.
                if (sd1 == INVALID_SOCKET)
                        return (0);


                iRLen = recv (wParam, (char FAR *) (sRcvBuff+ iRcvStart),  iRcvLen, 0);

                if (iRLen == SOCKET_ERROR)
                {

                        // Process error
                        return(0);
                }

                // Check if all the data was received
                if (iRLen== iRcvLen)
                {
                        // got all the data. Move it to output message buffer
                        // Get ready to receive another buffer

                }
                else    // still need more data to fill this buffer
                {
                        iRcvStart += iRLen;
                        iRcvLen -= iRLen;
                }
                return(0);

                ……
                ……
```