Unit OS A: Windows Networking

A.3. Microsoft-specific extensions to Sockets and other Networking APIs

Roadmap for Section A.3

- Windows Sockets (winsock2) Extensions
- Web access APIs
- Named pipes and mailslots
- NetBIOS / Wnet API

Winsock 2.0 Features

- Windows NT 4.0 was the first operating system with native support of the Winsock 2.0 specification:
  - Access to protocols other than TCP/IP,
  - Overlapped I/O,
  - Ability to query and request specific qualities of service
- New header file and libraries:
  - WS2_32.LIB / WS2_32.DLL
  - winsock2.h
- Microsoft extensions to Winsock have been moved out into their own separate DLL
  - WINSOCK.DLL contains forwarders to these routines

4

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>

5

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

6

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 = WSAGETSELECTERROR(lParam); (wError != 0 ?)

7

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 FD_WRITE events:

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

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

8

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.

9

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()

10

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

11

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.

12

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.

13

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()

16

Additional Asynchronous Socket Routines

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

17

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 completed */
   if(operation_complete()) break;
     /* normal completion */
}
```

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

18

Web API: Internet Support

- WININET.DLL supports HTTP, FTP, Gopher protocols
- General-purpose WinInet Functions
  - InternetOpen
  - InternetConnect
  - InternetOpenUrl
  - InternetReadFile
  - InternetCloseHandle
  - InternetSetStatusCallback
  - InternetQueryOption
  - InternetSetOption
  - InternetFindNextFile (FTP and Gopher)

19

Internet Suport (http/gopher protocols)

- WinInet HTTP Functions
  - HttpOpenRequest
  - HttpAddRequestHeaders
  - HttpSendRequest
  - HttpQueryInfo
- WinInet Gopher Functions
  - GopherFindFirstFile
  - GopherOpenFile
  - GopherCreateLocator
  - GopherGetAttribute

20

Internet Support (ftp protocol)

- WinInet FTP Functions
  - FtpFindFirstFile
  - FtpGetFile
  - FtpPutFile
  - FtpDeleteFile
  - FtpRenameFile
  - FtpOpenFile
  - InternetWriteFile
  - FtpCreateDirectory
  - FtpRemoveDirectory
  - FtpSetCurrentDirectory
  - FtpGetCurrentDirectory
  - FtpCommand
  - InternetGetLastResponseInfo

Most of the WinInet functions work with or return HINTERNETs (handles to Internet). While your code sees all HINTERNETs as the same type, one HINTERNET can mean something completely different from another.

The first type of HINTERNET that you obtain comes when you initialize WININET.DLL by calling InternetOpen. This is the first of 13 possible subtypes of HINTERNETs.

21

HINTERNET subtypes

INTERNET_HANDLE_TYPE_INTERNET	1
INTERNET_HANDLE_TYPE_CONNECT_FTP	2
INTERNET_HANDLE_TYPE_CONNECT_GOPHER	3
INTERNET_HANDLE_TYPE_CONNECT_HTTP	4
INTERNET_HANDLE_TYPE_FTP_FIND	5
INTERNET_HANDLE_TYPE_FTP_FIND_HTML	6
INTERNET_HANDLE_TYPE_FTP_FILE	7
INTERNET_HANDLE_TYPE_FTP_FILE_HTML	8
INTERNET_HANDLE_TYPE_GOPHER_FIND	9
INTERNET_HANDLE_TYPE_GOPHER_FIND_HTML	10
INTERNET_HANDLE_TYPE_GOPHER_FILE	11
INTERNET_HANDLE_TYPE_GOPHER_FILE_HTML	12
INTERNET_HANDLE_TYPE_HTTP_REQUEST	13

Query the subtype of a particular handle by calling:

InternetQuery Option

and sending it the HINTERNET with the INTERNET_OPTION_ HANDLE_TYPE parameter

22

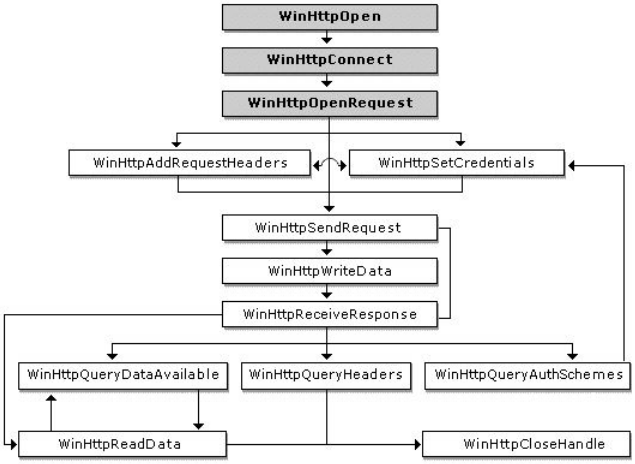
Windows HTTP Services (WinHTTP)

- HTTP client API to send requests through the HTTP protocol to other HTTP servers
  - WinHTTP supports desktop client applications, Windows services, and Windows server-based applications
  - WinHTTP offers both a C/C++ application programming interface (API) and a COM automation component use in Active Server Pages
- WinHTTP 5.1 is an OS component of:
  - Windows Server 2003 family
  - Windows XP SP1
  - Windows 2000 SP3 (except Datacenter Server)

23

Usage of the WinHTTP API

- Windows HTTP API exposes a set of C/C++ functions that enable applications to access HTTP resources on the Web



24

Using Internet Functions

- After setting up WININET.DLL, the initial HINTERNET is usually passed to InternetConnect,
  - parameter indicates the type of connecting server (HTTP, FTP, or Gopher).
  - InternetConnect returns another subtype of HINTERNET, which is then passed to the appropriate HTTP, FTP, or Gopher functions.
  - Alternatively call InternetOpenUrl (parses the URL; connects to the appropriate type of server automatically)
- After connecting to the desired server:
  - turn to the protocol-specific functions
  - read data from the server via InternetReadFile (works with HINTERNETs for any of the three supported protocols)
- Before exiting the program:
  - all of the HINTERNETs should be closed by calling InternetCloseHandle.

25

Named Pipes

- Message oriented:
  - Reading process can read varying-length messages precisely as sent by the writing process
- Bi-directional
  - Two processes can exchange messages over the same pipe
- Multiple, independent instances of a named pipe:
  - Several clients can communicate with a single server using the same instance
  - Server can respond to client using the same instance
- Pipe can be accessed over the network
  - location transparency
- Convenience and connection functions

Using Named Pipes

```
HANDLE CreateNamedPipe (LPCTSTR lpszPipeName,
        DWORD fdwOpenMode, DWORD fdwPipMode
        DWORD nMaxInstances, DWORD cbOutBuf,
        DWORD cbInBuf, DWORD dwTimeOut,
        LPSECURITY_ATTRIBUTES lpsa );
```

- lpszPipeName: \\.\pipe\[path]pipename
  - Not possible to create a pipe on remote machine (. – local machine)
- fdwOpenMode:
  - PIPE_ACCESS_DUPLEX, PIPE_ACCESS_INBOUND, PIPE_ACCESS_OUTBOUND
- fdwPipeMode:
  - PIPE_TYPE_BYTE or PIPE_TYPE_MESSAGE
  - PIPE_READMODE_BYTE or PIPE_READMODE_MESSAGE
  - PIPE_WAIT or PIPE_NOWAIT (will ReadFile block?)

Use same flag settings for all instances of a named pipe

Named Pipes (contd.)

- nMaxInstances:
  - Number of instances,
  - PIPE_UNLIMITED_INSTANCES: OS choice based on resources
- dwTimeOut
  - Default time-out period (in msec) for WaitNamedPipe()
- First CreateNamedPipe creates named pipe
  - Closing handle to last instance deletes named pipe
- Polling a pipe:
  - Nondestructive – is there a message waiting for ReadFile

```
BOOL PeekNamedPipe (HANDLE hPipe,
        LPVOID lpvBuffer, DWORD cbBuffer,
        LPDWORD lpcbRead, LPDWORD lpcbAvail,
        LPDWORD lpcbMessage);
```

28

Named Pipe Client Connections

- CreateFile with named pipe name:
  - \\.\pipe\[path]pipename
  - \\servername\pipe\[path]pipename
  - First method gives better performance (local server)
- Status Functions:
  - GetNamedPipeHandleState
  - SetNamedPipeHandleState
  - GetNamedPipeInfo

29

Convenience Functions

- WriteFile / ReadFile sequence:

```
BOOL TransactNamedPipe( HANDLE hNamedPipe,
        LPVOID lpvWriteBuf, DWORD cbWriteBuf,
        LPVOID lpvReadBuf, DWORD cbReadBuf,
        LPDOWRD lpcbRead, LPOVERLAPPED lpa);
```

- CreateFile / WriteFile / ReadFile / CloseHandle:
  - dwTimeOut: NMPWAIT_NOWAIT, NMPWAIT_WIAT_FOREVER, NMPWAIT_USE_DEFAULT_WAIT

```
BOOL CallNamedPipe( LPCTSTR lpszPipeName,
        LPVOID lpvWriteBuf, DWORD cbWriteBuf,
        LPVOID lpvReadBuf, DWORD cbReadBuf,
        LPDWORD lpcbRead, DWORD dwTimeOut);
```

30

Server: eliminate the polling loop

```
BOOL ConnectNamedPipe (HANDLE hNamedPipe,
        LPOVERLAPPED lpo );
```

- lpo == NULL:
  - Call will return as soon as there is a client connection
  - Returns false if client connected between CreateNamed Pipe call and ConnectNamedPipe()
- Use DisconnectNamedPipe to free the handle for connection from another client
- WaitNamedPipe():
  - Client may wait for server's ConnectNamedPipe()
- Security rights for named pipes:
  - GENERIC_READ, GENERIC_WRITE, SYNCHRONIZE

31

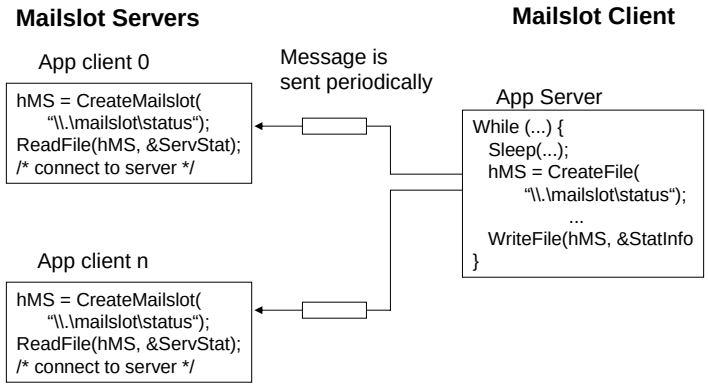
Windows IPC - Mailslots

Mailslots bear some nasty implementation details; they are almost never used

- Broadcast mechanism:
  - One-directional
  - Mutliple writers/multiple readers (frequently: one-to-many comm.)
  - Message delivery is unreliable
  - Can be located over a network domain
  - Message lengths are limited (w2k: < 426 byte)
- Operations on the mailslot:
  - Each reader (server) creates mailslot with CreateMailslot()
  - Write-only client opens mailslot with CreateFile() and uses WriteFile() – open will fail if there are no waiting readers
  - Client‘s message can be read by all servers (readers)
- Client lookup: *\mailslot\mailslotname
  - Client will connect to every server in network domain

32

Locate a server via mailslot



33

Creating a mailslot

```
HANDLE CreateMailslot(LPCTSTR lpszName,
        DWORD cbMaxMsg,
        DWORD dwReadTimeout,
        LPSECURITY_ATTRIBUTES lpsa);
```

- lpszName points to a name of the form
  - \\.\mailslot\[path]name
  - Name must be unique; mailslot is created locally
- cbMaxMsg is msg size in byte
- dwReadTimeout
  - Read operation will wait for so many msec
  - 0 – immediate return
  - MAILSLOT_WAIT_FOREVER – infinite wait

34

Opening a mailslot

- CreateFile with the following names:
  - \\.\mailslot\[path]name - retrieve handle for local mailslot
  - \\host\mailslot\[path]name - retrieve handle for mailslot on specified host
  - \\domain\mailslot\[path]name - returns handle representing all mailslots on machines in the domain
  - *\mailslot\[path]name - returns handle representing mailslots on machines in the system‘s primary domain: max mesg. len: 400 bytes
  - Client must specifiy FILE_SHARE_READ flag
- GetMailslotInfo() and SetMailslotInfo() are similar to their named pipe counterparts

35

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

36

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.

37

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.

38

Further Reading

- Mark E. Russinovich and David A. Solomon, Microsoft Windows Internals, 4th Edition, Microsoft Press, 2004.
  - Networking APIs (from pp. 791)
- Anthony Jones, Jim Ohmund, Jim Ohlund, James Ohlund, Network Programming for Microsoft Windows, 2nd Edition, Microsoft Press, 2002.
- Ralph Davis, Windows NT Network Programming, Addison-Wesley, 1996.

39