Lecture

WinSock Server Socket functions

```
int bind(
  SOCKET s,
  const struct sockaddr FAR *name,
  int namelen
);
struct sockaddr {
  u_short    sa_family;
  char       sa_data[14];
};
```

WinSock Server Socket functions

A special version of **`struct sockaddr`** is used for internet address familty **`AF_INET`**, and that defined as **`struct sockaddr_in`**

```
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

WinSock Client Socket functions

```
struct in_addr {
  union {
     struct { u_char s_b1,s_b2,s_b3,s_b4; }   S_un_b;
     struct { u_short s_w1,s_w2; }            S_un_w;
      u_long                                   S_addr;
  } S_un;
};
```

- Host and network byte-ordering:
  `htonl(), htons(), ntohl(), ntohs()`

Resolving Host Names

```
struct hostent FAR *gethostbyname(
  const char FAR *name
);
struct hostent {
  char FAR *        h_name;
  char FAR * FAR *  h_aliases;
  short             h_addrtype;
  short             h_length;
  char FAR * FAR *  h_addr_list;
};
```

WinSock Client Socket functions

```
int connect(
   SOCKET s,
   const struct sockaddr FAR *name,   SERVER's
   int namelen
);
```

- If socket *s*, is unbound, unique values are assigned to the local association by the system, and the socket is marked as bound.

Sending and receiving data from server

```
int send(
   SOCKET s,
   const char FAR *buf,
   int len,
   int flags
);
```

- Similarly **recv()** receives as much data as can fit in to the buffer. Returns # of bytes received.
- These are blocking calls, i.e. these functions do not return until data is sent/received

Difference between Server and Client socket calls

- Client: socket() > connect() > send/recv() > closesocket

- Server: socket() > bind() > listen() > accept() > recv/send () > closesocket

Listen

```
int listen( SOCKET s, int backlog );
```

Places a socket in a state where it is listening to an incoming connection and can accept it.

backlog specifies maximum number of pending connections; It can not be greater than **SO_MAXCON**

Returns 0 if all is well, otherwise **SOCKET_ERROR**

TIP: **SOCKET_ERROR** is in fact equal to -1

bind

```
int bind(
   SOCKET s,
   const struct sockaddr FAR *name,
   int namelen
);
```

associates a local address with a socket.

If no error, returns `0`, `SOCKET_ERROR` otherwise

Accept() *blocking call*

```
SOCKET accept(
  SOCKET s,
  struct sockaddr FAR *addr,
  int FAR *addrlen
);
```

Returns a **SOCKET** descriptor, or, if there is an error returns

INVALID_SOCKET

The **accept** function extracts the first connection on the queue of pending connections on socket *s*. It then creates a new socket and returns a handle to the new socket.

WinSock API calls

- The blocking problem
- Synchronous and Asynchronous I/O
- Non-blocking calls

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

- Advantage of original BSD socket calls over WinSock function calls

The blocking problem

- Resolving the blocking problem
- Multi-threaded solution
- What is a multi-threaded server?
- The need for synchronisation objects in multi-threaded server environments

Small WinSock application examples

- A client showing simple communication to either our own small server, or some server on the internet, e.g. WHOIS servers, HTTP server, time service etc.
- A small utility that synchronises system time with a source on the internet, accounting for transmission-delays

Introduction to Windows Sockets Programming

- WHOIS port 43
- WHOIS Server: whois.networksolutions.com
- Description of the functionality of this utility

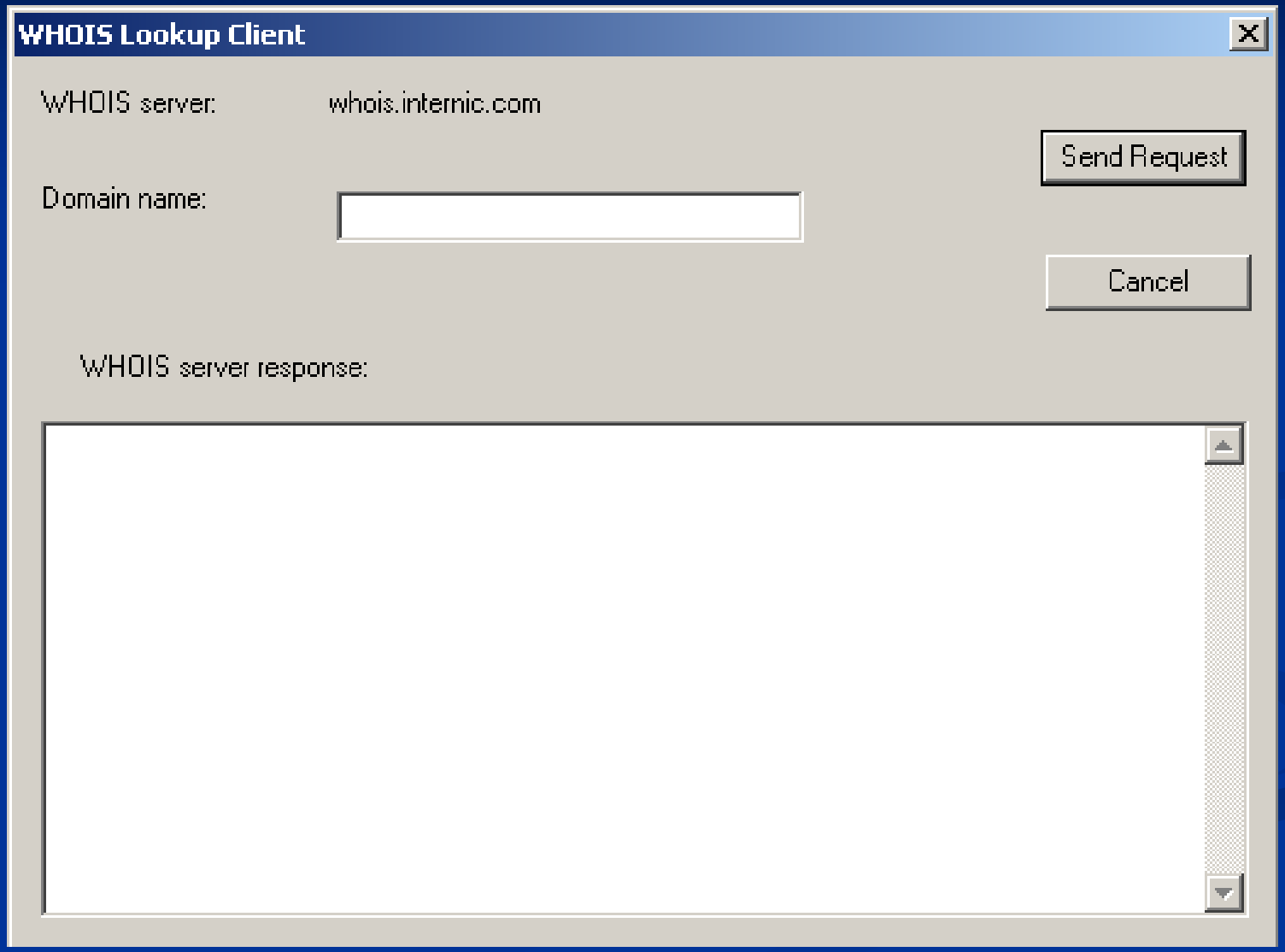
- Advantage of original BSD socket calls over WinSock function calls

Small WinSock application example

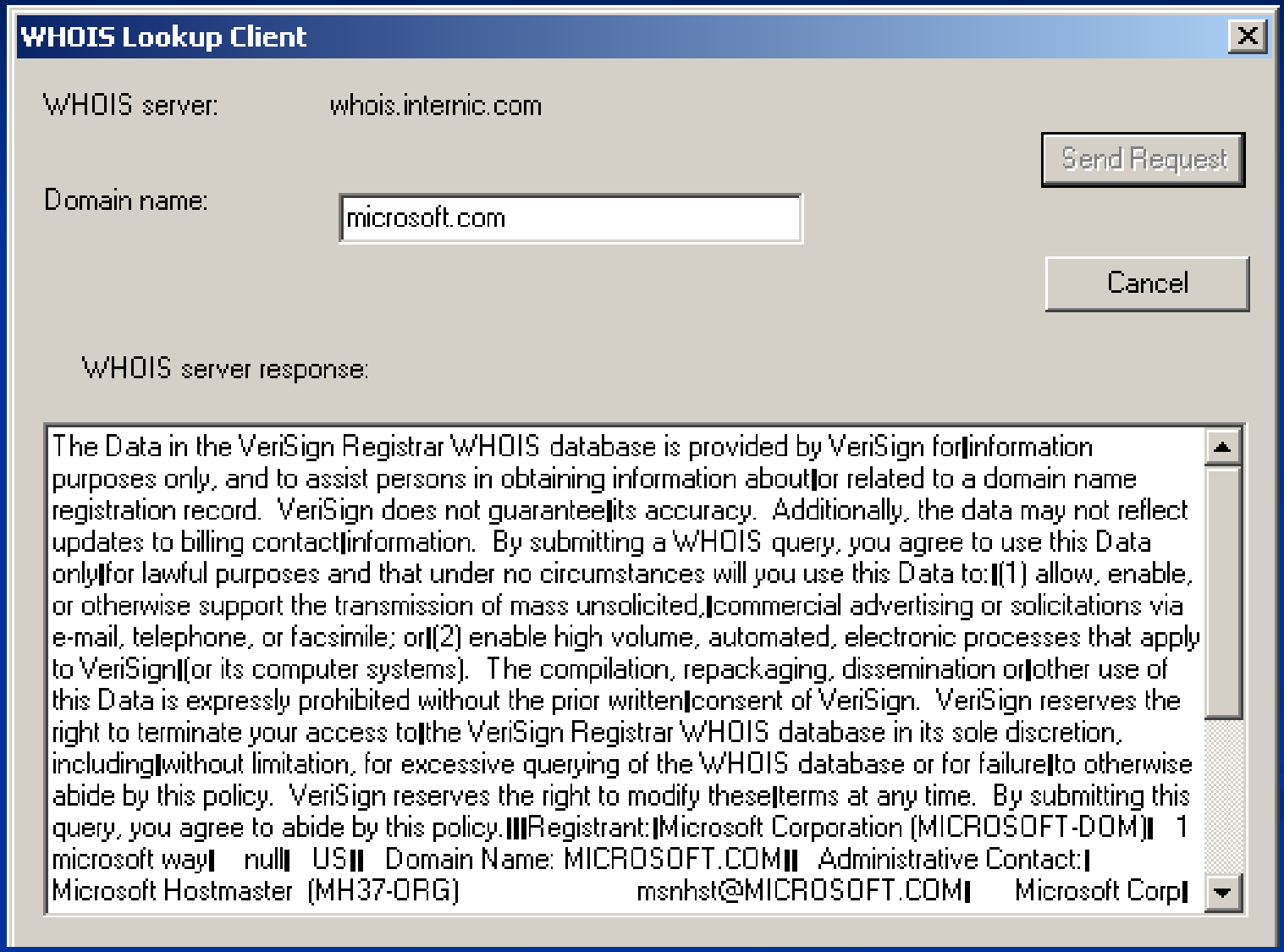
A WHOIS client with the following functions

- A dialog-based application
- Accepts a domain name from user
- connects to a WHOIS server on the internet, whose name is hard-coded in the application
- Sends a WHOIS request to the server
- Shows WHOIS server response in a multi-line edit control

Application User Interface



Application Output



application example

```
#define WHOIS_PORT                    43
#define WHOIS_SERVER_NAME
        "whois.networksolutions.com"
```

Other servers could be "whois.pknic.net.pk" or "whois.crsnic.net" etc.

```
#define MAX_DOMAIN_LEN                25
#define BUFFER_SIZE                   16384

BOOL CALLBACK mainDialogProc(HWND hDlg, UINT
    message, WPARAM wParam, LPARAM lParam);
SOCKET clientSocket;
```

application example

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE
    hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    WSADATA wsaData;
    HOSTENT *ptrHostEnt;
    struct sockaddr_in serverSockAddr;      // the address of the socket to
    connect to

    int abc;

    // try initialising the windows sockets library
    if(WSAStartup( MAKEWORD(1,1), &wsaData))// request WinSock ver
    1.1
    {
        MessageBox(NULL, "Error initialising sockets library.",
    "WinSock Error", MB_OK | MB_ICONSTOP);
        return 1;
    }
```

application example

```
if(!(ptrHostEnt = gethostbyname(WHOIS_SERVER_NAME)))
{
	MessageBox(NULL, "Could not resolve WHOIS server name.",
"WinSock Error", MB_OK | MB_ICONSTOP);
	WSACleanup();
	return 1;
}
```

application example

```
// fill out the address of the server.
    serverSockAddr.sin_family = AF_INET;     // fill the address structure with
    appropriate values
    serverSockAddr.sin_port = htons(WHOIS_PORT);   // MUST convert to
    network byte-order
    memset(serverSockAddr.sin_zero, 0, sizeof(serverSockAddr.sin_zero));
    memcpy(&serverSockAddr.sin_addr.S_un.S_addr, ptrHostEnt->h_addr_list[0],
    sizeof(unsigned long));

    clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(clientSocket == INVALID_SOCKET)
    {
        MessageBox(NULL, "Error creating client socket.", "WinSock Error",
    MB_OK | MB_ICONSTOP);
        WSACleanup();
        return 1;
    }
```

application example

```
if(connect(clientSocket, (struct sockaddr *)&serverSockAddr,
sizeof(serverSockAddr)))
{

        abc =  WSAGetLastError();

        MessageBox(NULL, "Error connecting to WHOIS server.", "WinSock
Error", MB_OK | MB_ICONSTOP);
        WSACleanup();
        return 1;
}

if(DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG_MAIN),
NULL, mainDialogProc) == 1)
        MessageBox(NULL, "Error occurred while sending data to WHOIS
server.", "WinSock Error", MB_OK | MB_ICONSTOP);

WSACleanup();

return 0;
}
```

application example

```
BOOL CALLBACK mainDialogProc(HWND hDlg, UINT message, WPARAM wParam,
    LPARAM lParam)
{
    int wID, wNotificationCode;
    char domainName[MAX_DOMAIN_LEN+2+1];  // accomodate CR/LF/NULL
    char result[BUFFER_SIZE], *startOfBuffer = result;
    int bytesReceived;

    switch(message)
    {
    case WM_INITDIALOG:
        SendDlgItemMessage(hDlg, IDC_EDIT_DOMAIN, EM_LIMITTEXT,
    MAX_DOMAIN_LEN, 0);
        return TRUE;
        break;

    case WM_COMMAND:
        wNotificationCode = HIWORD(wParam);
        wID = LOWORD(wParam);
        switch(wID)
        {
```

application example

```
case IDC_BUTTON_SEND:
                        EnableWindow(GetDlgItem(hDlg, IDC_BUTTON_SEND), FALSE);    // disable
    for 2nd use
                        GetDlgItemText(hDlg, IDC_EDIT_DOMAIN, (LPSTR)domainName,
    MAX_DOMAIN_LEN+1);
                        strcpy(domainName+strlen(domainName), "\r\n");
                        if(send(clientSocket, (const char *)domainName, strlen(domainName), 0) ==
    SOCKET_ERROR)
                                EndDialog(hDlg, 1);
                        else
                        {
                                bytesReceived = recv(clientSocket, startOfBuffer, BUFFER_SIZE-1, 0);
    // -1 for NULL
                                while(bytesReceived > 0)              // 0:close
    SOCKET_ERROR:error
                                {
                                        startOfBuffer += bytesReceived;      // move it forward
                                        bytesReceived = recv(clientSocket, startOfBuffer,
    BUFFER_SIZE-(startOfBuffer-result)-1, 0); // -1 for NULL
                                }
```

application example

```
                        if(startOfBuffer != result)          // something received
                          *startOfBuffer = NULL;             // NULL terminate
                        else
                                strcpy(result, "Null Response");

                        SetDlgItemText(hDlg, IDC_EDIT_RESULT, result);
                }
                break;

        case IDCANCEL:
                EndDialog(hDlg, 0);
                break;
        }
        return TRUE;
        break;

    default:
        return FALSE;
    }
    return TRUE;
}
```