Transitioning from UNIX to Windows Socket Programming

Paul O'Steen

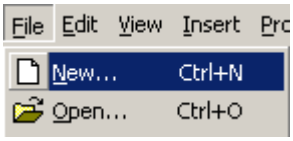
Most applications use Sockets to write application that communicate over the Internet; unfortunately, the Sockets interface varies slightly between operating systems. This guide provides brief instruction for those already familiar with UNIX Socket programming who wish to write Socket applications under Microsoft Windows using WinSock. To explain the differences between Sockets under UNIX and Windows, we demonstrate the adaptation of the TCP echo client presented in TCP/IP Sockets in C: Practical Guide for Programmers by Michael J. Donahoo and Kenneth L. Calvert. We begin by showing the setup of a Windows socket project in Microsoft Visual C++ 6.0 and the execution of a console application. Then we give a side-by-side comparison between the UNIX and Windows versions of the TCP echo client code, explaining each of the key differences. This comparison will demonstrate all of the key differences between UNIX and Windows Socket programming. It is important to understand that our objective is to make UNIX Socket application work under Windows using WinSock with as few changes as possible. Many other changes are possible to make the application take maximum advantage of WinSock specific features.

Creating a Windows Socket project

First, let's create a working version of the Windows adaptation of the TCP echo client. Download the adapted code from the book website at http://cs.baylor.edu/~donahoo/practical/CSockets. You will need at least TCPEchoClientWS.c and DieWithErrorWS.c. Now we can construct an executable using the Visual C++ development environment.

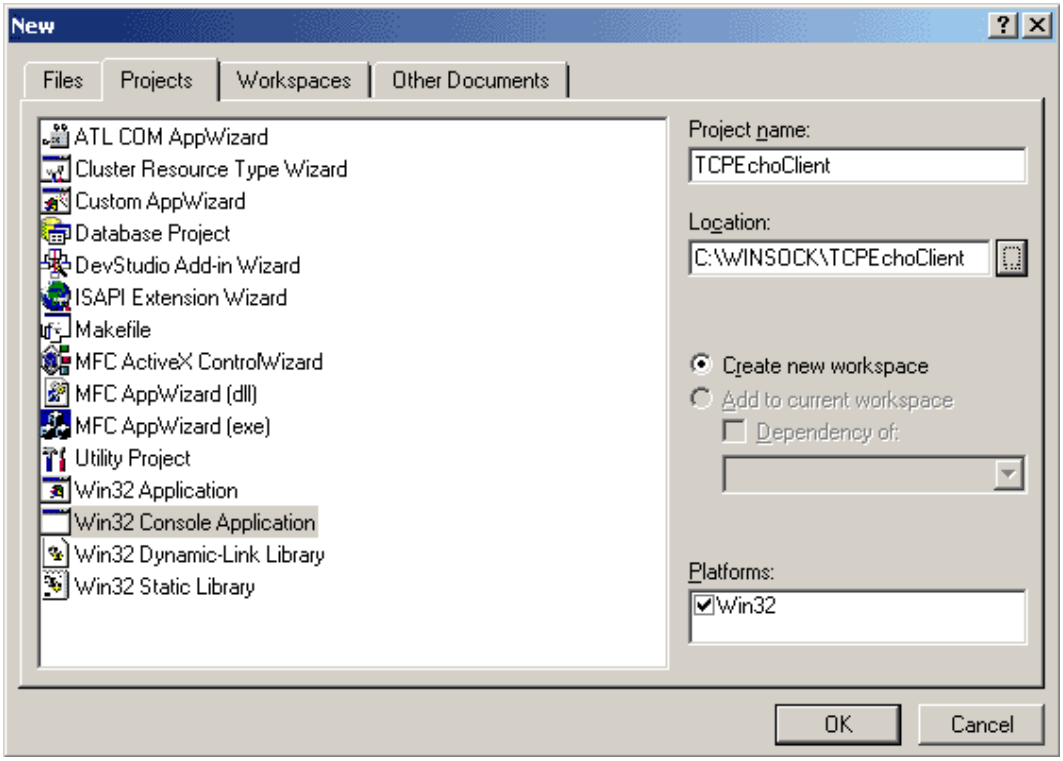
Making the Project

Setting up a project is the first step in working with Windows sockets. There are several important steps in creating a Windows socket project. First, open Microsoft Visual C++ 6.0 and click on *File* and then *New*.

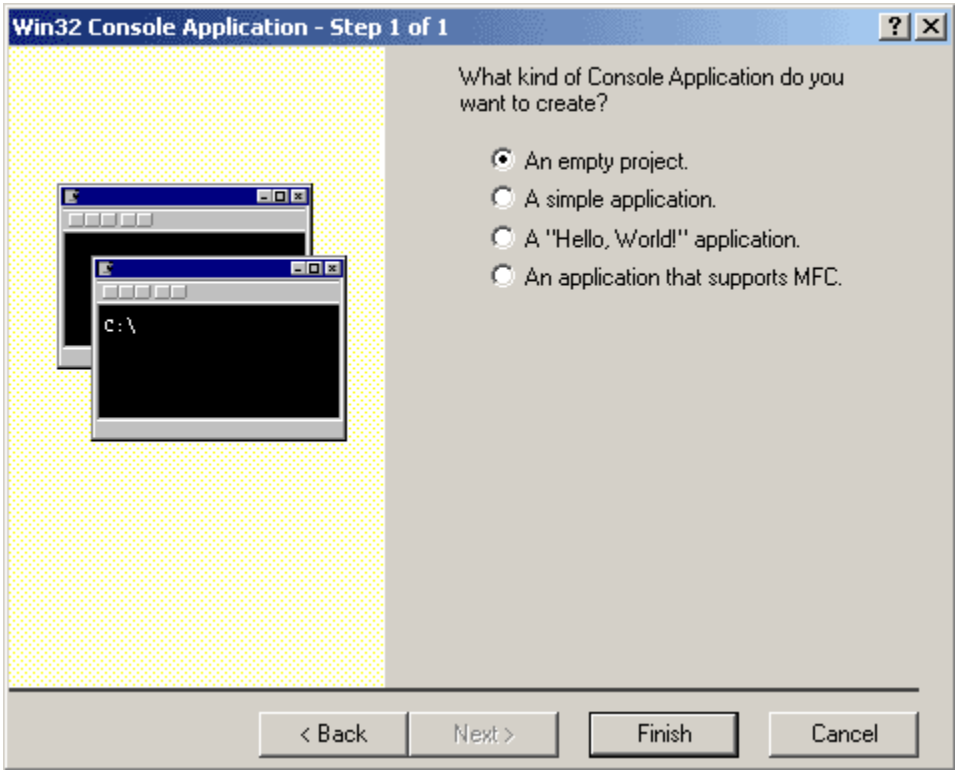


When the New window pops up, do the following:

1) select *Projects* tab
2) select *Win32 Console Application*
3) type a name in the project name field
4) pick a location to store your project
5) click *OK*.

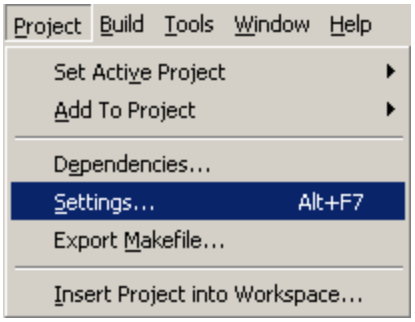


A window will now pop up and ask what kind of application you would like to create. Click on *An Empty Project* and then click *Finish*.

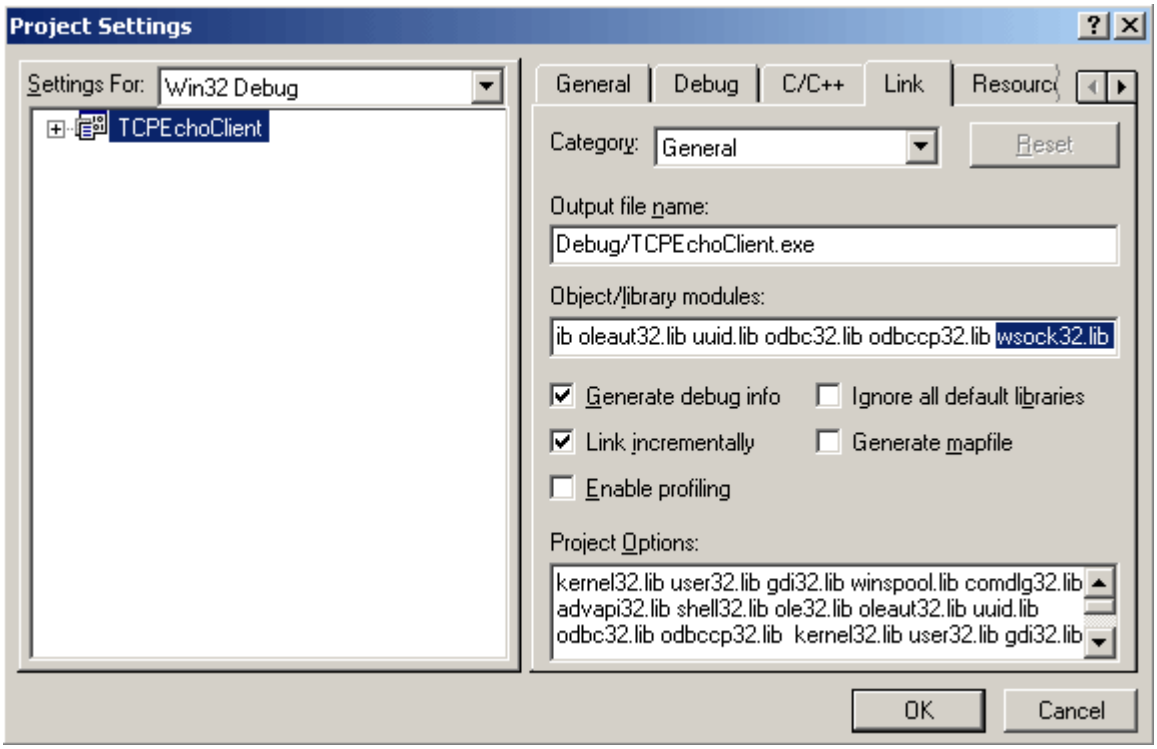


Adding Windows Sockets to the Project

Now click on *Project* and then *Settings*.



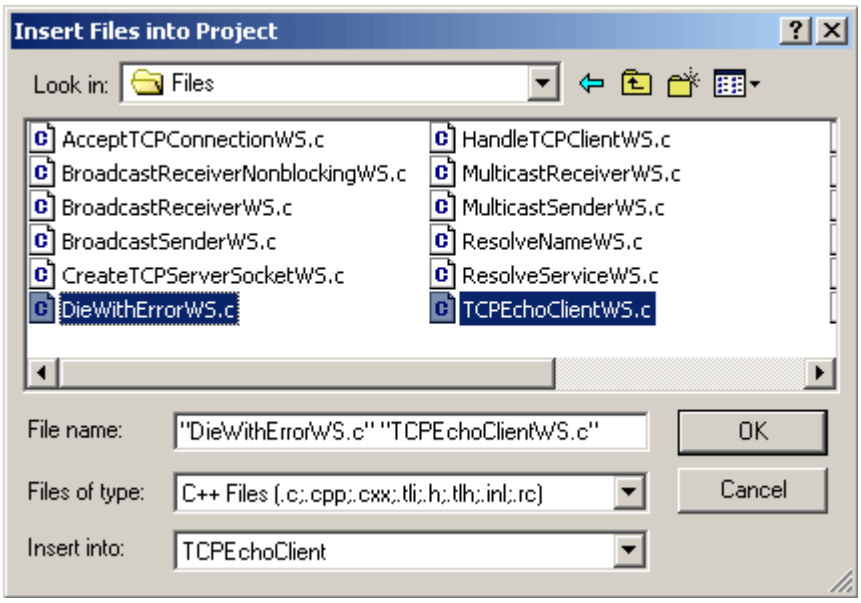
When the Settings window pops up, click on the *Link* tab in the top right corner. Under the *Object/libraries modules* heading, add *wsock32.dll* to the list and click *OK*.



Adding Files to the Project

Now it is time to add the source files that you will be using for your project. Click on *Project* and then *Add to Project* and then *Files*.

The *Insert Files* window will pop up and this is where you add the source file(s) to your project.
To add files to your project select the file or files you want to add and click *OK*. If you want to select multiple files, hold down the *Ctrl* key and click on the files you want.

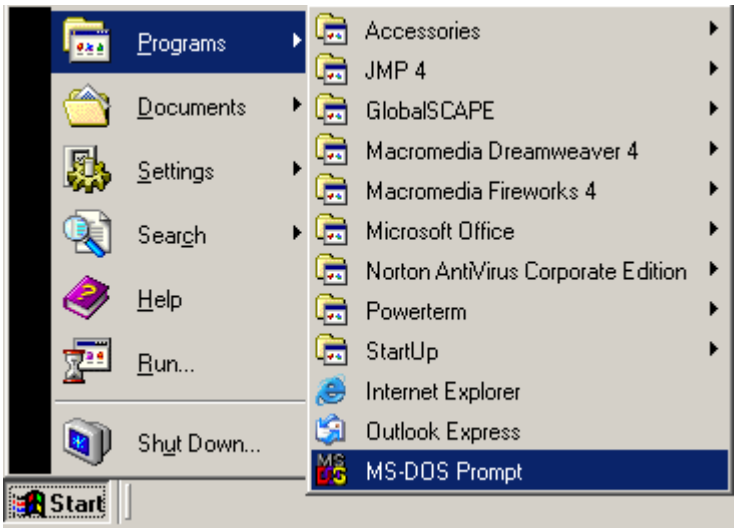


Using the Project from within Visual C++

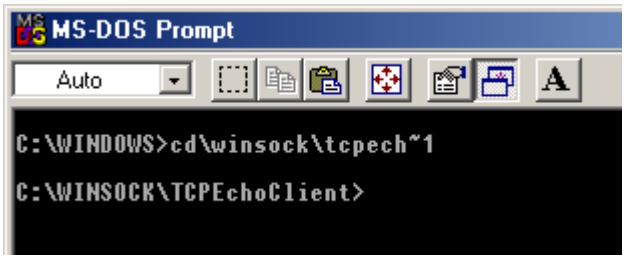
You may run your application from within Visual C++'s debug environment. First, we need to specify the parameters of our application to Visual C++. We do this under the Debug tab of the Project Settings dialog box. Select *Settings*... from the *Project* menu, click on the *Debug* tab, and enter the arguments in the *Program arguments:* field.)

Using the Project from the command line

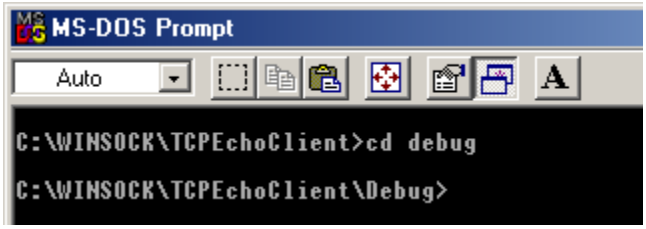
You can run applications from the command line by opening the MS-DOS prompt in Windows.



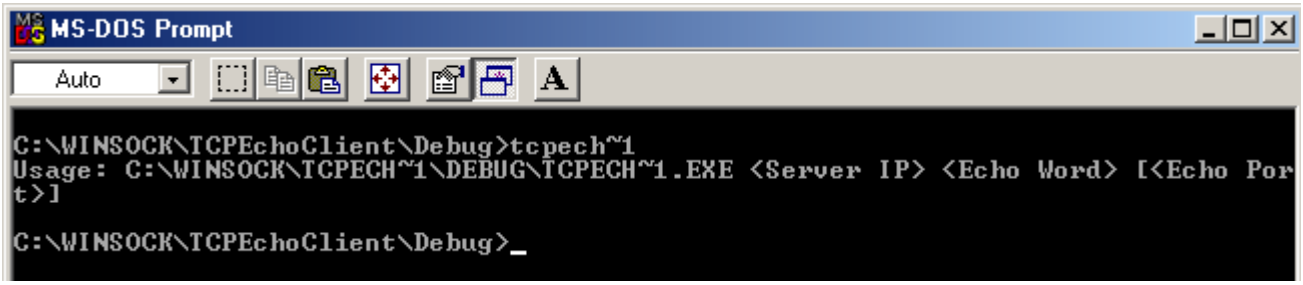
Once you have opened the MS-DOS prompt, you are ready to locate your file. When the project was created you selected the directory to store your project in the *Location:* and *Project Name* fields on the *New* window screen. To get to your project directory, you need to type *cd\directory* where directory is the *location\project name* of your project.



The executable is located in the Debug directory of the project directory.



To execute the application, type in the name of the file along with any arguments.



UNIX vs. Windows Sockets

To demonstrate the key differences, we perform a side-by-side comparison between the UNIX and adapted Windows Socket code. We break the programs into separate sections, each of which deals with a specific difference. Comments have been removed from the programs to allow focus on the key differences.

UNIX

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Windows

```
#include <stdio.h>
#include <winsock.h>
#include <stdlib.h>
```

Header File

Header files needed by Windows socket programs are slightly different than UNIX socket programs. For windows socket programming you need to add the *winsock.h* header file to your programs. Since this file includes all socket definitions and prototypes, several of the header files from the UNIX example are not needed.

UNIX

```
#define RCVBUFSIZE 32

void DieWithError(char *errorMessage);

int main(int argc, char *argv[])
{
  int sock;
  struct sockaddr_in echoServAddr;
  unsigned short echoServPort;
  char *servIP;
  char *echoString;
  char echoBuffer[RCVBUFSIZE];
  unsigned int echoStringLen;
  int bytesRcvd, totalBytesRcvd;


  if ((argc < 3) || (argc > 4))
  {
    fprintf(stderr, "Usage: %s
      <Server IP> <Echo Word>
      [<Echo Port>]\n",argv[0]);
    exit(1);
```

Windows

```
#define RCVBUFSIZE 32

void DieWithError(char *errorMessage);

void main(int argc, char *argv[])
{
  int sock;
  struct sockaddr_in echoServAddr;
  unsigned short echoServPort;
  char *servIP;
  char *echoString;
  char echoBuffer[RCVBUFSIZE];
  int echoStringLen;
  int bytesRcvd, totalBytesRcvd;
  WSADATA wsaData;

  if ((argc < 3) || (argc > 4))
  {
    fprintf(stderr, "Usage: %s
      <Server IP> <Echo Word>
      [<Echo Port>]\n",argv[0]);
    exit(1);
```

```
  }

  servIP = argv[1];
  echoString = argv[2];

  if (argc == 4)
    echoServPort = atoi(argv[3]);
  else
    echoServPort = 7;
```

```
  }

  servIP = argv[1];
  echoString = argv[2];

  if (argc == 4)
    echoServPort = atoi(argv[3]);
  else
    echoServPort = 7;

  if (WSAStartup(MAKEWORD(2, 0),
    &wsaData) != 0)
  {
    fprintf(stderr,"WSAStartup()
      failed");
    exit(1);
  }
```

Application Setup

The setup of the application is identical except for initialization code required by WinSock. The WinSock library is initialized by `WSAStartup( )`. The first parameter to the startup function is the version of WinSock the program wishes to use. We want version 2.0. `MAKEWORD(2, 0)` returns the version (2.0) number in the format expected by the startup function. The second parameter is a pointer to a WSADATA structure that allows the WinSock library to communicate critical information to the program such as limits on the number of sockets that can be created. `WSAStartup( )` fills in the WSADATA structure before returning.

```
typedef struct WSAData {
  WORD            wVersion;
  WORD            wHighVersion;
  char            szDescription[WSADESCRIPTION_LEN+1];
  char            szSystemStatus[WSASYS_STATUS_LEN+1];
  unsigned short  iMaxSockets;
  unsigned short  iMaxUdpDg;
  char FAR *      lpVendorInfo;
} WSADATA;
```

`WSAStartup( )` returns a zero on success or a non-zero number on failure.

UNIX and Windows

```
  if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    DieWithError("socket() failed");

  memset(&echoServAddr, 0, sizeof(echoServAddr));
  echoServAddr.sin_family      = AF_INET;
  echoServAddr.sin_addr.s_addr = inet_addr(servIP);
  echoServAddr.sin_port        = htons(echoServPort);

  if (connect(sock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
    DieWithError("connect() failed");

    echoStringLen = strlen(echoString);

  if (send(sock, echoString, echoStringLen, 0) != echoStringLen)
    DieWithError("send() sent a different number of bytes than expected");

  totalBytesRcvd = 0;
  printf("Received: ");
  while (totalBytesRcvd < echoStringLen)
  {
    if ((bytesRcvd = recv(sock, echoBuffer, RCVBUFSIZE - 1, 0)) <= 0)
      DieWithError("recv() failed or connection closed prematurely");
    totalBytesRcvd += bytesRcvd;
    echoBuffer[bytesRcvd] = '\0';
    printf(echoBuffer);
  }

  printf("\n");
```

Communication

The communication portion of the UNIX and Windows programs is identical.

UNIX

```
  closesocket(sock);
  WSACleanup();

  exit(0);
}
```

Windows

```
  close(sock);


  exit(0);
}
```

Application Shutdown

The shutdown of the application differs slightly between UNIX and Windows. Instead of `close( )`, WinSock uses `closesocket( )`; however, the two functions do the same thing so the difference is in name only. Finally, we must use WSACleanup( ) to deallocate the resources used by Winsock. `WSACleanup( )` returns a zero on success and a non-zero number on failure.

WinSock Error Reporting

Windows uses its own error message facility to indicate what went wrong with a Sockets call. `WSAGetLastError( )` returns an integer representing the last error that occurred. If a WinSock application needs to know why a socket call failed, it should call `WSAGetLastError( )` immediately after

the failed socket call. As you might expect, our error reporting function, `DieWithError( )` must be changed to work under Windows.

UNIX

```
#include <stdio.h>
#include <stdlib.h>

void DieWithError(char *errorMessage)
{
  perror(errorMessage);
  exit(1);
}
```

Windows

```
#include <stdio.h>
#include <winsock.h>
#include <stdlib.h>

void DieWithError(char *errorMessage)
{
  fprintf(stderr,"%s: %d\n",
    errorMessage, WSAGetLastError());
  exit(1);
}
```

`DieWithError( )`

The Windows version of `DieWithError( )` uses `WSAGetLastError( )` instead of perror to report system error messages.

Conclusion

Moving from UNIX sockets to Windows sockets is fairly simple. Windows programs require a different set of include files, need initialization and deallocation of WinSock resources, use `closesocket( )` instead of `close( )`, and use a different error reporting facility. However, the meat of the application is identical to UNIX.