

WINSOCK 2

WINDOWS SOCKET: PROGRAM EXAMPLES PART 11

Machine's OS is standalone Windows XP Pro with SP2 except whenever mentioned. Compiler used was Visual C++ 2003 .Net 1.1. Beware the codes that span more than one line. Program examples have been tested for Non Destructive Test. All information compiled for Windows 2000 (NT5.0) above and...

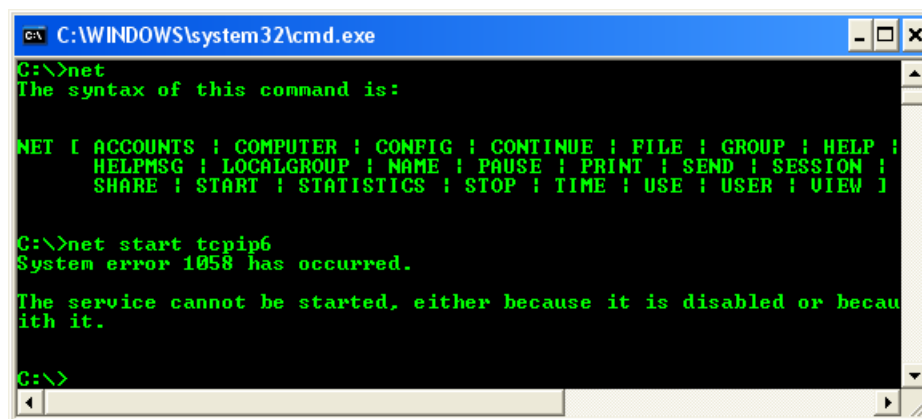
1. The story was discussed at [Winsock introduction story](#).
2. Related functions, structures and macros used in the program examples have been dumped at [Winsock structure & function 1](#) and [Winsock structure & function 2](#).
3. Other related and required information (if any) not available in no. 2 can be found at [MSDN & Visual C++ online reference](#).

Abilities

- Able to understand and use the IPv6.
- Able to understand Winsock implementation and operations of the IPv6 through the APIs and program examples.
- Able to gather, understand and use the Winsock [functions](#), [structures](#) and [macros](#) in your programs.
- Able to build programs that use Microsoft C/Standard C programming language and Winsock APIs.

Internet Protocol Version 6 (IPv6)

For Windows® Server 2003 family, Microsoft® provides support for the Internet Protocol version 6, IPv6. In my Windows Xp Pro machine there are IPv6 driver (**tcpip6.sys** driver file is present in the `%systemroot%\System32\Drivers` directory, but not installed by default. To check the IPv6 existence we can use the **net** command.



```
C:\WINDOWS\system32\cmd.exe
C:\>net
The syntax of this command is:

NET [ ACCOUNTS | COMPUTER | CONFIG | CONTINUE | FILE | GROUP | HELP |
HELPSMSG | LOCALGROUP | NAME | PAUSE | PRINT | SEND | SESSION |
SHARE | START | STATISTICS | STOP | TIME | USE | USER | VIEW ]

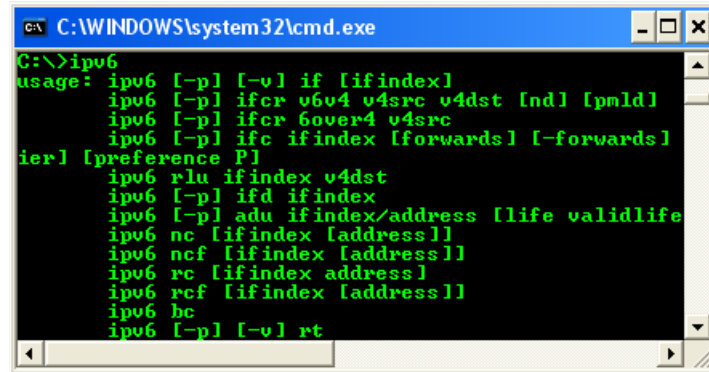
C:\>net start tcpip6
System error 1058 has occurred.

The service cannot be started, either because it is disabled or because
it has no enabled devices attached.

C:\>
```

Figure 1

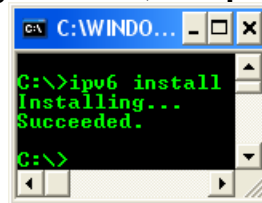
So the IPv6 driver is not installed. The system error 1058 - "The service cannot be started, either because it is disabled or because it has no enabled devices associated with it" - ERROR_SERVICE_DISABLED. We can use **ipv6** command to install, uninstall and configure the IPv6.



```
C:\WINDOWS\system32\cmd.exe
C:\>ipv6
usage: ipv6 [-p] [-v] if [ifindex]
ipv6 [-p] ifcr v6v4 v4src v4dst [nd] [pml]
ipv6 [-p] ifcr 6over4 v4src
ipv6 [-p] ifc ifindex [forwards] [-forwards]
ier] [preference P]
ipv6 rlu ifindex v4dst
ipv6 [-p] ifd ifindex
ipv6 [-p] adu ifindex/address [life validlife]
ipv6 nc [ifindex [address]]
ipv6 ncf [ifindex [address]]
ipv6 rc [ifindex address]
ipv6 rcf [ifindex [address]]
ipv6 hc
ipv6 [-p] [-v] rt
```

Figure 2

To install the IPv6 driver through console, run **ipv6 install**.



```
C:\WINDOWS...
C:\>ipv6 install
Installing...
Succeeded.
C:\>
```

Figure 3

Or you can use the **Local Area Network** properties page.

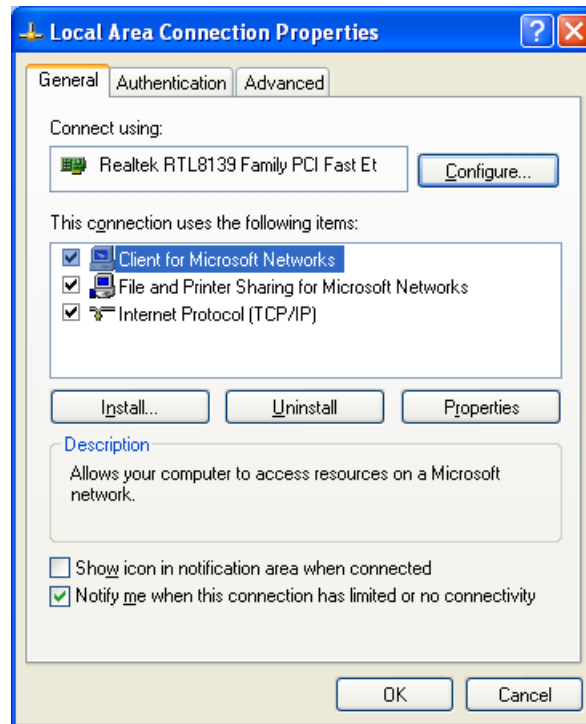


Figure 4

Click the **Install...** button. Then select the **Protocol** network component type and click the **Add...** button.

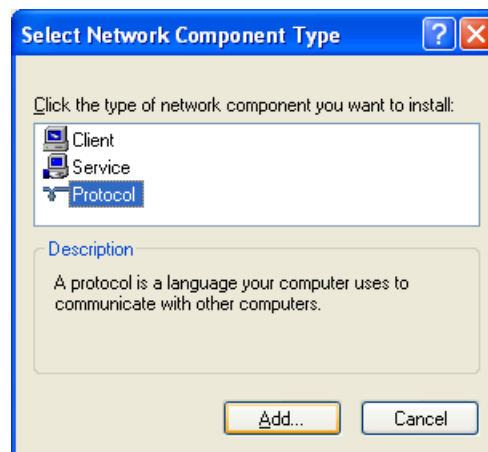


Figure 5

Select the **Microsoft TCP/IP version 6** protocol and click the **OK** button.

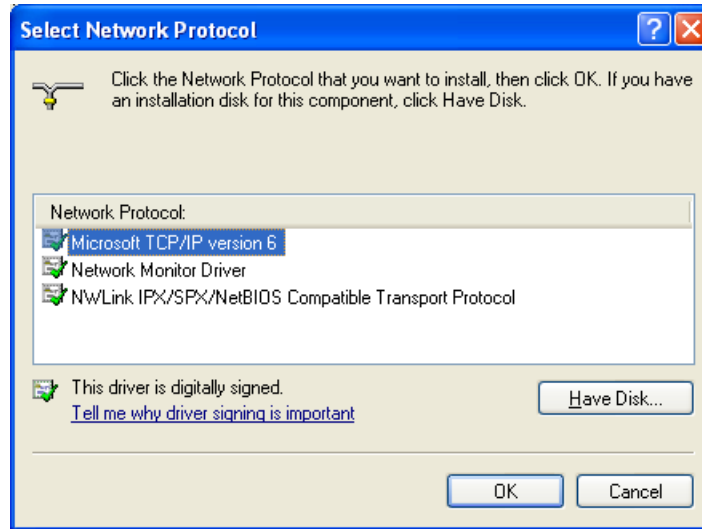


Figure 6

Then we start or stop the IPv6 service as other Windows services by running **net start tcpip6/net stop tcpip6** command.

```
C:\WINDOWS\system32\cmd.exe
C:\>net start tcpip6
The requested service has already been started.
More help is available by typing NET HELPMSG 2182.

C:\>net stop tcpip6
The following services are dependent on the Microsoft IPv6
Stopping the Microsoft IPv6 Protocol Driver service will al
IPv6 Helper Service
Do you want to continue this operation? <Y/N> [N]: y
The IPv6 Helper Service service was stopped successfully.
The Microsoft IPv6 Protocol Driver service is stopping....
The Microsoft IPv6 Protocol Driver service could not be sto
```

Figure 7

Then we can play around with IPv6 by using the **ipv6** command.

```
C:\WINDOWS\system32\cmd.exe
C:\>ipv6 if
Interface 5: Ethernet: Local Area Connection
Guid {52021CBE-7279-44DF-B7D3-16538E4E39C8}
cable unplugged
uses Neighbor Discovery
uses Router Discovery
link-layer address: 00-50-8d-e5-77-0d
tentative link-local fe80::250:8df:fee5:770d, life infinite
multicast interface-local ff01::1, 1 refs, not reportable
multicast link-local ff02::1, 1 refs, not reportable
multicast link-local ff02::1:ffe5:770d, 1 refs, last reporter
```

Figure 8

You can also check the IPv6 setting by using the **ipconfig** command. The following is a client-server program example for IPv6 enabled machines. The first one is the server program example run for IPv4. If you are in IPv6 network environment, please try running the program using the IPv6 address and PF_INET6 family.

// IPv6 server program example. Try running it on the IPv6 enabled machines using IPv6 arguments

```
#define WIN32_LEAN_AND_MEAN
#include <winsock2.h>
#include <ws2tcpip.h>
#ifndef IPPROTO_IPV6
// For IPv6.
#include <tcpv6.h>
#endif
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

// This code assumes that at the transport level, the system only supports one stream protocol (TCP) and one datagram protocol (UDP). Therefore, // specifying a socket type of SOCK_STREAM is equivalent to specifying TCP and specifying a socket type of // SOCK_DGRAM is equivalent to specifying UDP.

```
// Accept either IPv4 or IPv6
#define DEFAULT_FAMILY PF_UNSPEC
// TCP socket type
#define DEFAULT_SOCKETTYPE SOCK_STREAM
// Arbitrary, test port
#define DEFAULT_PORT "2007"
// Set very small for demonstration purposes
#define BUFFER_SIZE 128
```

```
void Usage(char *ProgName)
{
    fprintf(stderr, "\nSimple socket server program.\n");
    fprintf(stderr, "\n%s [-f family] [-t transport] [-p port] [-a address]\n\n",
ProgName);
    fprintf(stderr, " family\tOne of PF_INET, PF_INET6 or PF_UNSPEC. (default %s)\n",
        (DEFAULT_FAMILY == PF_UNSPEC) ? "PF_UNSPEC" :
        ((DEFAULT_FAMILY == PF_INET) ? "PF_INET" : "PF_INET6"));
    fprintf(stderr, " transport\tEither TCP or UDP. (default: %s)\n",
        (DEFAULT_SOCKETTYPE == SOCK_STREAM) ? "TCP" : "UDP");
}
```

```

    fprintf(stderr, " port\t\tPort on which to bind. (default %s)\n",
DEFAULT_PORT);
    fprintf(stderr, " address\tIP address on which to bind. (default: unspecified
address)\n");
    WSACleanup();
    exit(1);
}

```

```

LPSTR DecodeError(int ErrorCode)

```

```

{
    static char Message[1024];

    // If this program was multi-threaded, we'd want to use
    FORMAT_MESSAGE_ALLOCATE_BUFFER instead of a static buffer here.
    // (And of course, free the buffer when we were done with it)
    FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM |
    FORMAT_MESSAGE_IGNORE_INSERTS |
                FORMAT_MESSAGE_MAX_WIDTH_MASK, NULL, ErrorCode,
                MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
(LPSTR)Message, 1024, NULL);
    return Message;
}

```

```

int main(int argc, char **argv)

```

```

{
    char Buffer[BUFFER_SIZE], Hostname[NI_MAXHOST];
    int Family = DEFAULT_FAMILY;
    int SocketType = DEFAULT_SOCKETYPE;
    char *Port = DEFAULT_PORT;
    char *Address = NULL;
    int i, NumSocks, RetVal, FromLen, AmountRead;
    SOCKADDR_STORAGE From;
    WSADATA wsaData;
    ADDRINFO Hints, *AddrInfo, *AI;
    SOCKET ServSock[FD_SETSIZE];
    fd_set SockSet;

    printf("Usage: %s [-f family] [-t transport] [-p port] [-a address]\n", argv[0]);
    printf("Example: %s -f PF_INET6 -t TCP -p 1234 -a 127.0.0.1\n", argv[0]);
    printf("Else, default values used. By the way, the -a not usable for
server...\n");
    printf("Ctrl + C to terminate the server program.\n\n");

    // Parse arguments
    if (argc > 1)
    {
        for(i = 1; i < argc; i++)

```

```

{
  if ((argv[i][0] == '-') || (argv[i][0] == '/') &&
      (argv[i][1] != 0) && (argv[i][2] == 0))
  {
    switch(tolower(argv[i][1]))
    {
      case 'f':
        if (!argv[i+1])
          Usage(argv[0]);
        if (!strcmp(argv[i+1], "PF_INET"))
          Family = PF_INET;
        else if (!strcmp(argv[i+1], "PF_INET6"))
          Family = PF_INET6;
        else if (!strcmp(argv[i+1], "PF_UNSPEC"))
          Family = PF_UNSPEC;
        else
          Usage(argv[0]);
        i++;
        break;

      case 't':
        if (!argv[i+1])
          Usage(argv[0]);
        if (!strcmp(argv[i+1], "TCP"))
          SocketType = SOCK_STREAM;
        else if (!strcmp(argv[i+1], "UDP"))
          SocketType = SOCK_DGRAM;
        else
          Usage(argv[0]);
        i++;
        break;

      case 'a':
        if (argv[i+1])
        {
          if (argv[i+1][0] != '-')
          {
            Address = argv[++i];
            break;
          }
        }
        Usage(argv[0]);
        break;

      case 'p':
        if (argv[i+1])
        {

```

```

        if (argv[i+1][0] != '-')
        {
            Port = argv[++i];
            break;
        }
    }
    Usage(argv[0]);
    break;

default:
    Usage(argv[0]);
    break;
}
} else
    Usage(argv[0]);
}
}

// Ask for Winsock version 2.2...
if ((RetVal = WSASStartup(MAKEWORD(2, 2), &wsaData)) != 0)
{
    fprintf(stderr, "Server: WSASStartup() failed with error %d: %s\n", RetVal,
DecodeError(RetVal));
    WSACleanup();
    return -1;
}
else
    printf("Server: WSASStartup() is OK.\n");

if (Port == NULL)
{
    Usage(argv[0]);
}

// By setting the AI_PASSIVE flag in the hints to getaddrinfo(), we're indicating
that we intend to use the resulting address(es) to bind
// to a socket(s) for accepting incoming connections. This means that when
the Address parameter is NULL, getaddrinfo() will return one
// entry per allowed protocol family containing the unspecified address for that
family.
memset(&Hints, 0, sizeof(Hints));
Hints.ai_family = Family;
Hints.ai_socktype = SocketType;
Hints.ai_flags = AI_NUMERICHOST | AI_PASSIVE;
RetVal = getaddrinfo(Address, Port, &Hints, &AddrInfo);
if (RetVal != 0)
{

```

```

    fprintf(stderr, "getaddrinfo() failed with error %d: %s\n", RetVal,
gai_strerror(RetVal));
    WSACleanup();
    return -1;
}
else
    printf("Server: getaddrinfo() is OK.\n");

// For each address getaddrinfo returned, we create a new socket, bind that
address to it, and create a queue to listen on.
for (i = 0, AI = AddrInfo; AI != NULL; AI = AI->ai_next, i++)
{
    // Highly unlikely, but check anyway.
    if (i == FD_SETSIZE)
    {
        printf("Server: getaddrinfo() returned more addresses than we could
use.\n");
        break;
    }

    // This example only supports PF_INET and PF_INET6.
    if ((AI->ai_family != PF_INET) && (AI->ai_family != PF_INET6))
        continue;

    // Open a socket with the correct address family for this address.
    ServSock[i] = socket(AI->ai_family, AI->ai_socktype, AI->ai_protocol);
    if (ServSock[i] == INVALID_SOCKET)
    {
        fprintf(stderr, "Server: socket() failed with error %d: %s\n",
WSAGetLastError(), DecodeError(WSAGetLastError()));
        continue;
    }
    else
        printf("Server: socket() is OK.\n");

    // bind() associates a local address and port combination with the socket just
created. This is most useful when
    // the application is a server that has a well-known port that clients know
about in advance.
    if (bind(ServSock[i], AI->ai_addr, int(AI->ai_addrlen)) == SOCKET_ERROR)
    {
        fprintf(stderr, "Server: bind() failed with error %d: %s\n",
WSAGetLastError(), DecodeError(WSAGetLastError()));
        continue;
    }
    else
        printf("Server: bind() is OK.\n");
}

```

```

    // So far, everything we did was applicable to TCP as well as UDP.
    // However, there are certain fundamental differences between stream
    // protocols such as TCP and datagram protocols such as UDP.
    //
    // Only connection orientated sockets, for example those of type
    SOCK_STREAM, can listen() for incoming connections.
    if (SocketType == SOCK_STREAM)
    {
        if (listen(ServSock[i], 5) == SOCKET_ERROR)
        {
            fprintf(stderr, "Server: listen() failed with error %d: %s\n",
WSAGetLastError(), DecodeError(WSAGetLastError()));
            continue;
        }
        else
            printf("Server: listen() is OK.\n");
    }

    printf("I'm listening and waiting on port %s, protocol %s, protocol family
%s\n",
        Port, (SocketType == SOCK_STREAM) ? "TCP" : "UDP",
        (AI->ai_family == PF_INET) ? "PF_INET" : "PF_INET6");
}

freeaddrinfo(AddrInfo);

if (i == 0)
{
    fprintf(stderr, "Fatal error: unable to serve on any address.\n");
    WSACleanup();
    return -1;
}
NumSocks = i;

// We now put the server into an eternal loop, serving requests as they arrive.
FD_ZERO(&SockSet);
while(1)
{
    FromLen = sizeof(From);

    // For connection orientated protocols, we will handle the packets
    comprising a connection collectively. For datagram
    // protocols, we have to handle each datagram individually.
    //
    // Check to see if we have any sockets remaining to be served from
    previous time through this loop. If not, call select()

```

```

// to wait for a connection request or a datagram to arrive.
for (i = 0; i < NumSocks; i++)
{
    if (FD_ISSET(ServSock[i], &SockSet))
        break;
}
if (i == NumSocks)
{
    for (i = 0; i < NumSocks; i++)
        FD_SET(ServSock[i], &SockSet);
    if (select(NumSocks, &SockSet, 0, 0, 0) == SOCKET_ERROR)
    {
        fprintf(stderr, "Server: select() failed with error %d: %s\n",
WSAGetLastError(), DecodeError(WSAGetLastError()));
        WSACleanup();
        return -1;
    }
    else
        printf("Server: select() is OK.\n");
}
for (i = 0; i < NumSocks; i++)
{
    if (FD_ISSET(ServSock[i], &SockSet))
    {
        FD_CLR(ServSock[i], &SockSet);
        break;
    }
}

if (SocketType == SOCK_STREAM)
{
    SOCKET ConnSock;

    // Since this socket was returned by the select(), we know we have a
    connection waiting and that this accept() won't block.
    ConnSock = accept(ServSock[i], (LPSOCKADDR)&From, &FromLen);
    if (ConnSock == INVALID_SOCKET)
    {
        fprintf(stderr, "Server: accept() failed with error %d: %s\n",
WSAGetLastError(), DecodeError(WSAGetLastError()));
        WSACleanup();
        return -1;
    }
    else
        printf("Server: accept() is OK.\n");
}

```

```

    if (getnameinfo((LPSOCKADDR)&From, FromLen, Hostname,
sizeof(Hostname), NULL, 0, NI_NUMERICHOST) != 0)
        strcpy(Hostname, "<unknown>");
    printf("\nAccepted connection from %s.\n", Hostname);

    // This sample server only handles connections sequentially. To handle
multiple connections simultaneously, a server
    // would likely want to launch another thread or process at this point to
handle each individual connection. Alternatively,
    // it could keep a socket per connection and use select() on the fd_set to
determine which to read from next.
    //
    // Here we just loop until this connection terminates.
    while (1)
    {
        // We now read in data from the client. Because TCP does NOT
maintain message boundaries, we may recv()
        // the client's data grouped differently than it was sent. Since all this
server does is echo the data it
        // receives back to the client, we don't need to concern ourselves about
message boundaries. But it does mean
        // that the message data we print for a particular recv() below may
contain more or less data than was contained
        // in a particular client send().
        AmountRead = recv(ConnSock, Buffer, sizeof(Buffer), 0);
        if (AmountRead == SOCKET_ERROR)
        {
            fprintf(stderr, "Server: recv() failed with error %d: %s\n",
WSAGetLastError(), DecodeError(WSAGetLastError()));
            closesocket(ConnSock);
            break;
        }
        else
            printf("Server: recv() is OK.\n");

        if (AmountRead == 0)
        {
            printf("Server: Client closed the connection.\n");
            closesocket(ConnSock);
            break;
        }

        printf("Server: Received %d bytes from client: \"%.*s\"\n",
AmountRead, AmountRead, Buffer);
        printf("Server: Echoing the same data back to client...\n");

        RetVal = send(ConnSock, Buffer, AmountRead, 0);

```

```

        if (RetVal == SOCKET_ERROR)
        {
            fprintf(stderr, "Server: send() failed: error %d: %s\n",
WSAGetLastError(), DecodeError(WSAGetLastError()));
            closesocket(ConnSock);
            break;
        }
        else
            printf("Server: send() is OK.\n");
    }

    else
    {
        // Since UDP maintains message boundaries, the amount of data
        // we get from a recvfrom() should match exactly the amount of data the
        // client sent in the corresponding sendto().
        AmountRead = recvfrom(ServSock[i], Buffer, sizeof(Buffer), 0,
(LPSOCKADDR)&From, &FromLen);
        if (AmountRead == SOCKET_ERROR)
        {
            fprintf(stderr, "Server: recvfrom() failed with error %d: %s\n",
WSAGetLastError(), DecodeError(WSAGetLastError()));
            closesocket(ServSock[i]);
            break;
        }
        else
            printf("Server: recvfrom() is OK.\n");

        if (AmountRead == 0)
        {
            // This should never happen on an unconnected socket, but...
            printf("Server: recvfrom() returned zero, aborting...\n");
            closesocket(ServSock[i]);
            break;
        }
        else
            printf("Server: recvfrom() is OK, returning non-zero.\n");

        RetVal = getnameinfo((LPSOCKADDR)&From, FromLen, Hostname,
sizeof(Hostname), NULL, 0, NI_NUMERICHOST);
        if (RetVal != 0)
        {
            fprintf(stderr, "Server: getnameinfo() failed with error %d: %s\n",
RetVal, DecodeError(RetVal));
            strcpy(Hostname, "<unknown>");
        }
    }
}

```

```

else
    printf("Server: getnameinfo() is OK.\n");

    printf("Server: Received a %d byte datagram from %s: \"%%.*s\"\n",
AmountRead, Hostname, AmountRead, Buffer);
    printf("Server: Echoing the same data back to client\n");

    RetVal = sendto(ServSock[i], Buffer, AmountRead, 0,
(LPSOCKADDR)&From, FromLen);
    if (RetVal == SOCKET_ERROR)
    {
        fprintf(stderr, "Server: sendto() failed with error %d: %s\n",
WSAGetLastError(), DecodeError(WSAGetLastError()));
    }
    else
        printf("Server: sendto() is OK.\n");
}
}

return 0;
}

```

```

C:\WINDOWS\system32\cmd.exe - mywinsock -f PF_INET -t TCP -p 12345
C:\>mywinsock -f
Usage: mywinsock [-f family] [-t transport] [-p port] [-a address]
Example: mywinsock -f PF_INET6 -t TCP -p 1234 -a 127.0.0.1
Else, default values used. By the way, the -a not usable for server...
Ctrl + C to terminate the server program.

Simple socket server program.

mywinsock [-f family] [-t transport] [-p port] [-a address]

family      One of PF_INET, PF_INET6 or PF_UNSPEC. (default PF_UNSPEC)
transport   Either TCP or UDP. (default: TCP)
port        Port on which to bind. (default 2007)
address     IP address on which to bind. (default: unspecified address)

C:\>mywinsock -f PF_INET -t TCP -p 12345
Usage: mywinsock [-f family] [-t transport] [-p port] [-a address]
Example: mywinsock -f PF_INET6 -t TCP -p 1234 -a 127.0.0.1
Else, default values used. By the way, the -a not usable for server...
Ctrl + C to terminate the server program.

Server: WSStartup() is OK.
Server: getaddrinfo() is OK.
Server: socket() is OK.
Server: bind() is OK.
Server: listen() is OK.
I'm listening and waiting on port 12345, protocol TCP, protocol family PF_INET

```

Figure 9