Embedded Netsock™ - A DOS-Compatible UDP/IP Stack

The Goal

In the desktop PC world, networking is the big black hole. Servers, clients, protocols, services, adapters, addresses, etc. Highly trained network administrators spend weeks trying to get functions to work. This network administration work is challenging - and frustrating. But the benefits of networked desktop PCs are so great, that, to date, this large expense of time and money has just been accepted.

But, as corporations bemoan this significant expense, Microsoft and others have tried to address the underlying issues. Some aspects of recent releases of Windows NT are driven by the recognition that this expense is becoming exorbitant.

But don't be fooled into complacency - desktop networks currently demand extensive setup and configuration. You may reduce it, but it just doesn't seem to go away.

Embedded systems, while needing to tap into the vast resource of desktop PC networking technology, cannot have on-staff networking experts ship with each unit. These computers are buried into systems without an operator, and the networking ***must*** be made to require less setup and configuration. Neither the designer, the installer, nor the user of an embedded system should be forced to be a networking expert. So, what's to be done?

The Micro/sys Embedded Netsock Concept

The key to simplifying embedded network setup, configuration, and programming is to limit the networking universe to a subset that is relevant to embedded systems.

Here's what years of embedded networking experience at Micro/sys have brought us to do:

- Use a subset of the world-wide standard TCP/IP protocol suite for open architecture and support
- Use a subset of the popular Winsock API functions so that programmers have support
- Assume that shrink-wrap software is probably not applicable and that embedded system networking will probably incur custom programming, so make that programming as easy as possible
- Assume that all embedded PCs will be attached to the same, in-facility Ethernet network so that administrative issues such as routing are minimized
- Assume that a Windows™ desktop PC will be on the same Ethernet network as the networked embedded computers, and that it will probably serve as a supervisory computer
- Assume that, if needed, inter-facility networking will be provided by the Windows desktops, so that embedded PCs won't be burdened with Wide Area Network (WAN) issues
- If Windows NT Server is used as the supervisory computer, use the built-in, well-documented network administration tools such as DHCP to ease installation and operation
- Pre-install all layers of software on the embedded PC so that, at run time, the embedded application program can call a network function to perform a network operation, and everything from that software call to the bits on the wire is "pre-integrated" and transparent to the programmer

Micro/sys' Embedded Netsock Implementation

To achieve these goals, Micro/sys has implemented a complete, turnkey embedded network solution based on our Embedded Netsock firmware. Shipped pre-installed and pre-integrated on a Micro/sys embedded PC, Embedded Netsock makes creating embedded network applications much easier.

Here are some of the Embedded Netsock implementation details:

On a Micro/sys embedded PC that has an Ethernet port, a streamlined UDP/IP “stack” is included in the on-board firmware. This UDP/IP stack is located in on-board flash memory, and is available for use upon power-up. The programmer is presented with a subset of the Winsock API in order to create network-based application programs for the embedded PC. This API is well known and documented, and is ideal for use as a higher level network programming model.

There are no libraries to link into an application program. Merely including the NETSOCK.H include file into a C program will provide access at run-time to all Embedded Netsock functions. The UDP/IP stack is dynamically linked into the application program at run-time startup.

The Micro/sys Flash Setup™ firmware, which is also pre-installed on the embedded PC, provides the ability to do simple, straightforward configuration of each embedded PC's network subsystem. For instance, you can enter a network address into the embedded computer, or alternatively, indicate that the embedded computer is to obtain a network address, when it powers up, from another computer on the network using the Dynamic Host Configuration Protocol (DHCP).

Windows 95, 98, and NT are all supported as the supervisory computer on the Ethernet network. Custom programs written in Visual C++ or Visual BASIC can immediately network with embedded PCs running Embedded Netsock. Windows NT Server is, however, the most secure, robust, and well supported supervisory computer. Although not required, it can make management of embedded networks easier with such features as DHCP services.

Networking Support for Embedded Applications at Run-time

Once an embedded computer is up and running, the application program has access to the network through Winsock “sockets”. A socket is merely an abstraction for a network endpoint. You figuratively “plug one end of a wire into the socket on the embedded computer, and the other end of the wire into a socket on another computer (either the Windows desktop or another embedded computer)”.

In reality, of course, the actual wiring already ties all computers on the network together. Again, the “sockets” are merely an abstraction that allows a simple model for the software to use.

The Winsock specification defines three types of sockets:

SOCK_DGRAM sockets for “one time” packets from point A to point B, called “datagrams”
SOCK_RAW sockets for “low-level” network use, such as troubleshooting
SOCK_STREAM sockets for “open connections”, full-duplex transfers between point A and point B

To reduce size, complexity, and cost, the Micro/sys Embedded Netsock implementation supports datagrams and low-level sockets, but not “open connection” sockets. This is consistent with the needs of most all embedded applications. And it offers significant reductions in the setup and configuration aspects of the embedded network, as well as reducing memory size and CPU requirements on each embedded computer.

Selected TCP/IP Protocols

The TCP/IP family of protocols operates on the model of “layers”. Each layer is another layer of software that has its own specific tasks to perform in the overall networking scheme. A set of these software layers is called a “TCP/IP stack”, as one layer is stacked on top of the next.

The streamlined TCP/IP stack that is a key component in Embedded Netsock implements a number of protocols, and omits many that do not relate to embedded systems. Some poetic license has been taken in terms of the division of tasks between layers in this implementation. The goal is a small, embedded system oriented stack, not academic purity.

Here is a run-down on our streamlined TCP/IP stack, from the “bottom” up:

At the ***physical*** layer, 10BASE-T (twisted pair) or 10BASE-2 (thin coax) Ethernet are supported, depending upon the selected Micro/sys embedded PC. A custom software driver for the Ethernet adapter is included as part of the stack.

At the ***datalink*** layer, Address Resolution Protocol (ARP) is provided to cross reference Internet Protocol (IP) addresses to Ethernet hardware addresses. If the application program wants to send a datagram to a specific IP address, ARP is used to determine what Ethernet address must be used to perform this transfer.

At the ***network*** level, Internet Protocol (IP) is used to address datagrams from one computer to another. IP addresses are the now familiar “dotted decimal” numbers such as 207.217.56.206 (which happens to be the Micro/sys web site at embeddedsys.com). Internet Control Message Protocol (ICMP) is also provided at the network level for troubleshooting (i.e. PING utilities).

At the ***transport*** level, User Datagram Protocol (UDP) is supported. UDP is the basis upon which Winsock SOCK_DGRAM sockets are built. It is a one-way, single event type of protocol. Any IP address can send a UDP datagram to any other IP address. In official network terminology, UDP is classified as an "unreliable" protocol. (So is the IP protocol, for that matter.) This merely means that the sending station does not get packet-by-packet acknowledgements from the *protocol stack* of the receiving station. Simple acknowledgements from the *application program* can greatly increase reliability. Full CRC error checking is provided on each UDP packet, and with the Embedded Netsock assumption that all embedded PCs are on the same Ethernet wiring, the possibility of a UDP packet not being delivered is almost zero. The other common transport protocol, Transport Control Protocol (TCP) is not implemented due to its size and configuration complexity.

At the ***session*** level, various structures are implemented to maintain information about the local embedded PC. IP addresses and similar information is kept. For network initialization, the Dynamic Host Configuration Protocol (DHCP) is supported. DHCP allows an embedded PC to have almost no network configuration prior to startup. By contacting a “DHCP server” at start-up time, the embedded PC can request an IP address, and other network-oriented parameters. This allows centralized control, at the Windows NT server, of the embedded PC network.

At the Winsock API level, support is provided for standard UDP-oriented Winsock calls such as socket(), bind(), sendto(), and recvfrom().

Winsock UDP API

Here is a typical snippet of code that uses the standard Winsock API to send a datagram containing "Hello, world" from the local embedded computer to port number 5001 on remote computer 192.168.1.54.

```
#include "netsock.h"

SOCKET sock;
char string[80] = {"Hello, world"};
struct sockaddr_in local, dest;
int ret;

ret = WSAStartup();
if(ret = = SOCKET_ERROR)
    { handle error }

sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
if(sock = = INVALID_SOCKET)
    { handle error }

local.sin_family = AF_INET;
local.sin_port = htons(5001);
ret = bind(sock, &local, sizeof(local));
if(ret = = SOCKET_ERROR)
    { handle error }

dest.sin_family = AF_INET;
dest.sin_port = htons(5001);
dest.sin_addr.s_addr = inet_addr("192.168.1.54");

ret = sendto(sock, string, strlen(string), 0, dest, sizeof(dest));
if(ret = = SOCKET_ERROR)
    { handle error }
```

Conclusion

The technologies for efficient, effective embedded networking are readily available. By carefully paring down the possibilities to a subset that is relevant to embedded systems, design, implementation, installation, and administration can all be greatly simplified.

Embedded Netsock technology from Micro/sys brings a new level of quick startup and ease of application development to embedded networking. As an example, the Netsock/100 product is a PC/104 embedded PC that includes the full Embedded Netsock system ready to go, for a very affordable price. There are no royalties for the on-board operating system or Embedded Netsock, making it ideal for OEM design-ins.

Micro/sys offers the Embedded Netsock stack on 188, 366, 485, and Pentium embedded PCs.

If full networking capabilities, including FTP, HTTP, etc. are needed, consider Linux, VxWorks™, or one of the other 32-bit operating systems that are available. Contact Micro/sys Tech Sales for details.