Fundamentals of NetBIOS and LANtastic Networks

Performance Criteria

After completing this module, you will be able to do the following:

+ Explain how LANtastic uses NetBIOS for network communications.
+ Explain how LANtastic servers and workstations share files and printers.
+ Trace a packet of information from an application on a workstation to a hard drive on a server.
+ Identify important NetBIOS commands and explain their use.
+ Define major terms in LANtastic network communications.

The Plan

In this module, we'll do the following:

+ Trace data as it passes between Redirector and Server.
+ Discuss the operation of NetBIOS and how it's utilized in a LANtastic network.
+ Examine the role of each driver as it packages and moves data.
+ Discuss various settings for the network drivers and their effect on network operation and performance.

Network Operating Systems

DEFINITION

Let's get the boring stuff out of the way first by coming up with a definition for a Network Operating System (NOS).

> In general, a NOS encompasses protocols, specifications, interfaces, and support programs that establish and control communications between computers.

This is a broad definition, though, that would include unnetwork-like activities like file transfers via modem (eg, Procomm) or serial cable (eg, Laplink), or transactions between dumb terminals and mainframes or minicomputers.

These programs lack a key ingredient commonly associated with a network: transparent operation. Using LapLink or Procomm to transfer files constrains you, more or less, to that particular interface.

A NOS, on the other hand, becomes part of the background, like a fawn in the forest.

> From a hardware viewpoint, a NOS becomes an extension of the computer's operating system. It takes the disk storage and printing services provided by the OS on a local machine and makes them available to other machines with little or no intervention by the user.
>
> Joe can type a report on his computer...

...using a program he accessed from Cynthia's computer

...print it on a laserjet connected to Angela's computer

...save it to a drive on Adam's computer.

The computers might not be in the same building, or even the same continent. For the time being, communications are limited to this sector of the galaxy, but hey, it's only a matter of time, right?

DISTINCTIONS

Networked computers, like most social institutions, divide into two groups: the haves and the have-nots:

+ The haves are called Servers. Servers possess *resources* such as drives, printers, CD-ROMs, modems, and other peripherals that it offers to the network for other machines to use.

+ The have-nots are called clients (Novell) or workstations (Good-guys). Workstations have drives and printers, too, but they keep them to themselves.

Karl Marx would have loved networks had they been around a century ago. Servers give to each workstation according to its ability; workstations take from each server according to their need.

As you might expect, given the history of this kind of unbalanced sharing, there are difficulties to be overcome:

+ Servers have to be powerful, fast, and capable of prolonged operation free from lockups and other annoying habits, like smoking.

+ Workstations, for their part, must obey certain rules of etiquette to keep from taking up all of the server's time.

So-called "client/server" network operating systems rigorously enforce the distinction between servers and workstations. Servers serve. Workstations work. Period. Liza Dolittle never gets to dance with a prince no matter how well she pronounces Hartford, Herriford, and Hampshire. These oligarchical NOSs even bring in thugs called "system administrators" to police the boundaries.

LANtastic, on the other hand, uses a peer-to-peer networking model that more closely resembles the way people like to live and do business.

+ Machines can be servers and workstations at the same time.

+ Users can be working away at their computers doing all sorts of useful things while sharing each others drives and printers transparently and easily.

+ If someone needs a little privacy, all they have to do is stop sharing their resources, as easy as flicking a light switch.

NOS MODELS

Computers communicate in much the same way as humans. Both transactions are heavily larded with customs and ceremonies and manners:

You and I meet for the first time.

We make eye contact.

We introduce ourselves.

We shake hands.

We ask about spouses and jobs and home towns.

We exchange business cards.

Eventually, we break off our conversation and go off on other business.

For computers, transactions like this are govered by protocols and standards.

Failure to follow these protocols cause the same result for computers as it does for people...you either get no communication at all or the message gets garbled.

The exact nature of the conversation between networked machines is covered in the section titled "Outside the Black Box -- Network Messaging."

EXAMPLES OF NETWORKS, VENDORS, AND PROTOCOLS

Vendor	NOS	Protocol
Artisoft	LANtastic	Server Network Block (SNB)
Novell	Netware	Netware Core Protocol (NCP)
IBM	OS/2 LAN Server	Server Message Block (SMB)
Microsoft	Win 4 Wkgrps and NT	Server Message Block (SMB)
UNIX	SCO Linux	Network File System (NFS)

REVIEW OF NETWORKING STANDARDS

A wide variety of societies, oversight committees, and vendor groups determine how local area networks are constructed, designed, and interconnected. Only a couple of these standards have any real bearing on our present discussion.

In the 1980s, local area networking was at the same stage as automobile manufacturing in the early part of the century. No two car models could accept parts from each other. About the only standard was that there was no standard.

The International Open Systems foundation tried to bring some order to the LAN mayhem by promulgating a conceptual model for fashioning a network. This model is known as the Open Systems Interface (OSI) model.

All vendors give a passing nod to the OSI model but it's doubtful that Engineering groups stay up nights trying to ensure their code meets the letter and spirit of the model. Completely compatible open systems are, after all, invitations for competition.

The Institute of Electrical and Electronic Engineers (IEEE) brought it's weight to bear on this issue and had more success getting vendors to comply with their specifications, mostly because it adopted the LAN methodologies of IBM, DEC, and Xerox. Something like getting the Capones to clean up Chicago.

The IEEE defined interfaces between network modules and left the vendors free to do pretty much as they pleased within those interfaces.

Module HDW02, *Fundamentals of Ethernet Communications,* contains details of the OSI model, IEEE packet specifications, and network architectures. The examples in this module assume a LANtastic ethernet network using coax cable.

NETWORK HARDWARE AND SOFTWARE

Before two machines can talk to each other, they need hardware to convert their internal musings into a form that can successfully make the trip across a network cable. The examples in this module assume a single-segment ethernet network using Artisoft Noderunner adapters connected together with coax cable.

The hardware requires drivers and the NOS requires communication software. Several programs implement these features in LANtastic. The operation of each will be discussed in excrutiating detail in this module.

ADAPTER DRIVER -- handles low level data transfer to and from the network adapter. Noderunner adapters use either NODERUN.EXE or NR.EXE depending on the style of card. (NR2000SI cards use NR.EXE.)

TRANSPORT DRIVER -- takes data from an application on one machine and packages it for delivery by the adapter. LANtastic uses AILANBIO.EXE for this function.

NETWORK INTERFACE -- provides a set of controls that give access to the underlying network transport protocols. LANtastic uses the NetBIOS interface. NetBIOS stands for Network Basic Input Output System.

This difference can be confusing because LANtastic uses the AILANBIO driver as both a NetBIOS transport driver and a NetBIOS interface.

Third-party Networks And Their Transport Protocols

NetBIOS is an application programming interface. This API can be used as a front end to a variety of transport protocols. These include:

Network	Transport
Win for Wkgrps & NT	NETBEUI
OS/2 LAN Server	NETBEUI (same name, different program)
Internet	TCP/IP (Transport Control Protocol/ Internet Protocol)
Netware	IPX (Internetwork Packet Exchange)

REDIRECTOR -- nabs local file and print requests that involve the network and redirects them to the appropriate network server. LANtastic, along with other programs like Windows for Workgroups, uses REDIR.EXE (pronounced 'reader') for this purpose. REDIR is a NetBIOS program.

SERVER -- makes file and printer services available to other machines on the network. In this way, a server's hard drive can appear to be a local drive on a workstation. SERVER.EXE is the NetBIOS server program in LANtastic.

Redirector and Server work together like a tiny military detachment. Lieutenant Redirector scribbles down orders and Sergeant Server gives snappy acknowledgements then tries its best to get everything done as quickly as possible. The other programs and drivers exist merely to facilitate their communication.

A peer NOS such as LANtastic doesn't own the machine on which it runs. The various NOS components such as Redirector and Server and AILANBIO rent slices of the CPU's time like Minnesota tourists in a Florida condominium with DOS acting as landlord.

In some ways, DOS is a gracious host.

+ It gives the network access to just about all the machine's hardware like floppy and hard drives, printer ports, serial ports, and even the keyboard and video display if it's asked properly.
+ It's reasonable about working with network adapter cards as long as they keep to their own IRQ and IOBASE address and RAM address and don't cause a fuss if the CPU can't pay constant attention to them.
+ It also is willing to step aside for protected-mode applications while still keeping the real-mode applications happy by letting them run in the background.

In other ways, though, DOS can be downright cantankerous.

+ It was designed as an operating system for a 64 Kbyte machine with two floppy drives and a monochrome monitor. Don't be fooled by the fact that it can handle advanced application software with reasonable speed. It's a dancing bear and no mistake.
+ It does one thing at a time. Period. If the network needs to handle multiple tasks, it has to find ways to get this work done while giving run time to the other applications that constantly vie for DOS's attention.
+ It has a limited memory addressing space, forcing the network drivers to be as trim as possible, which limits their functionality.
+ It uses convoluted methods for going about its business, methods the network must respect even if they result in slow performance.

The network avoids many of the vagarities of DOS by appealing directly to the CPU. This isn't necessarily easier, but it's much faster.

The CPU is a busy critter, though. It has more than just the network to deal with. It has to watch over all the system peripherals like the timer, clock, monitor, disk drives, memory refresh circuits, etc etc. It also has to service any of a host of applications and TSRs that are loaded and running in memory.

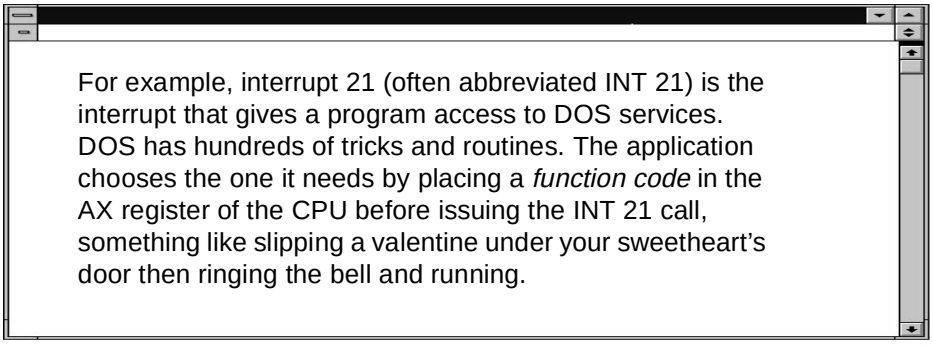
INTERRUPTS

Reading this section is not, strictly speaking, necessary for understanding networks and NetBIOS. The network relies heavily on DOS, it's generally a good idea

The inside of a computer is a rowdy environment. Like a stern schoolmarm, the CPU brings order to the chaos by enforcing a "raised hand" rule. If an application or a peripheral needs the CPU's attention, it issues an *interrupt*.

+ Programs issue *software interrupts*. "Please, CPU, have Mr. DOS open this particular file and tell him to give me the first 200 bytes, if you would be so kind."
+ Peripherals and adapter cards issue *hardware interrupts*. "Please, CPU, take this byte from my registers and hand it to the program that's waiting for it."

The Intel architecture provides a smörgåsbord of 256 interrupts, many with subfunctions.



Here's an example of the assembly language code that tells DOS to fess up the version of itself that's currently running:

```
MOV  AX, 4452 // moves function 44, subfunction 52 into the AX register
INT 21        // issues interrupt 21, the DOS general service interrupt
```

If a program issued these commands, it would then have to fetch the result from the appropriate CPU register and put it into a useful format.

WHO'S REALLY IN CHARGE?

The CPU owns the machine. Think of DOS as a butler that waits quietly until the Master calls.

In the old 50s Sherlock Holmes movies, all the British manors had bell pulls in each room for summoning a servant. A rope from each bell pull snaked back to the pantry where it connected to a bell dangling from a steel coil. When the master (or mistress) of the manor yanked a bell pull, the butler in the pantry would hear the tinkle and look up to see which coil was bouncing. He would then hurry to the specified room with the appropriate item: whisky for the billiard room, biscuits to the parlor, soap to the bath, etc.

The CPU reacts in a similar fashion to an interrupt.

+ Each interrupt has a set of instructions, called *Interrupt Service Routines (ISRs.)* These ISRs are also called *handlers.* ISRs, or handlers, tell the computer exactly what to do when an interrupt occurs.
+ Some handlers are tailored for the specific hardware of the machine. The instructions for these handlers are hard-coded in ROM BIOS.
+ Some handlers are contained in IO.SYS and MSDOS.SYS, DOS's two kernel files that load when the computer boots.
+ Still others are put in place by device drivers and TSRs. This makes the computer highly flexible and eliminates the need for swapping out chips or changing DIP switches or resoldering connections when a function changes.

Interrupts and their handlers comprise some of the most important code in a computer, much more important than the applications running in the foreground. After all, an amnesiac might not recall her name but she can still sip water from a glass. A computer without interrupt handlers, on the other hand, is about as capable of useful work as a beached jellyfish.

In recognition of their special status, the machine gives interrupt handlers priority treatment. One of the first things the bootstrap loader does when the machine is turned on or reset is set aside the first 1024 bytes of main memory, from hex 00000 to 00400, for storing the addresses of the interrupt handlers.

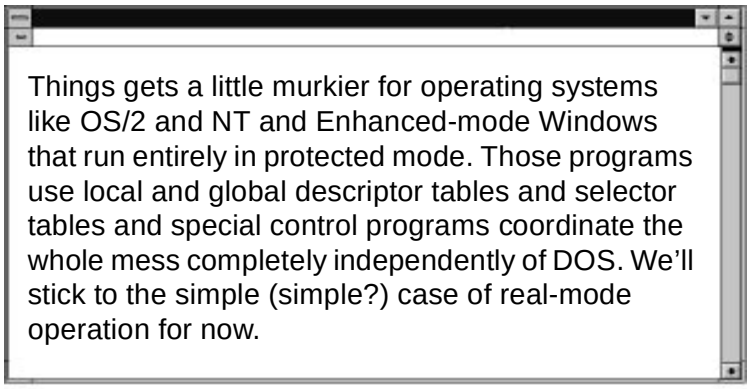
The array that holds this information is called an *Interrupt Vector Table (IVT.)*

> **What's a vector?** If you're travelling to your new job in Joliet, IL, and I tell you that Joliet lies southwest of Chicago along Interstate 80, I've given you a vector to your job.

How Interrupts Work

This section will give you some additional details about the operation of interrupts. You can skip it and still understand the remaining topics.

This business of interrupts and handlers and vector tables sounds complex, but the actual implementation is relatively trivial, something like a scavenger hunt.



When a program issues an interrupt, it tells the CPU, in essence, "Go to the Interrupt Vector Table, find the location of the handler, go to that location and carry out the instructions."

But what if the CPU is busy working on another instruction when it gets the interrupt? That's not only likely, it's darn well a certainty. The CPU has a host of housekeeping chores to do even when nothing is apparently going on.

The CPU has no choice in the matter. It must pay attention to the interrupt. So, if it's busy, it does the same thing you do when the phone rings while you're watching a video. You put the VCR on *pause* and pick up the phone, and when you're done with your conversation, you start the VCR and continue watching the movie.

When an application issues an interrupt:

+ The CPU shoves the instructions that currently occupy its attention into a special place in memory called a *stack*.
+ It takes a peek at the Interrupt Vector Table to find the location of the handler for the function number in its AX register.
+ It jumps to that location in memory.
+ It begins toiling away at its new task.
+ The last instruction in the handler's code tells the CPU to return to its knitting. This is called an IRET, for Instruction RETurn.

+ When the CPU gets to the IRET, it goes to the stack and fetches the information it left there.
+ It then continues on with the first set of instructions as if nothing had happened.

But, you ask, what if a second interrupt comes in while the CPU is working on the first one?

Not a problem. The CPU patiently pushes the current instruction from the first handler into the stack and starts servicing the new interrupt. And if a third interrupt arrives, the CPU pushes the second aside in the same manner.

A busy CPU stacks up interrupt after interrupt, one after another, like John Henry laying out railroad ties. It makes no decision about priorities. It does whatever it was last told to do.

INTERRUPT EXAMPLE USING DEBUG

The computer has a system clock. DOS has an INT 21 function, function 2A, that fetches the time from that clock. That sounds simple enough.

From the command prompt, run DEBUG. You'll get the Debug "-" (dash) prompt.

Type A and press *Enter*. The A stands for Assemble.

> Debug will give you the segment:offset address where it's holding your code. All the commands you enter from this point on will be stored sequentially in memory.

Type MOV AH,2A and press *Enter*. This stores the function code 2A in the high byte (AH) of the AX register.

> Note that the address advances two bytes. That's one 16-bit word. If that doesn't make any sense at all, don't worry about it. It's best not to try and second-guess Debug. It can leave a nasy bite. (Sorry.)

Type INT 21 and press *Enter.* This is the interrupt.

> Nothing happens! Don't worry. Debug won't run the code until you tell it to do so. Note that you advance another two bytes in memory.

Type INT 3 and press *Enter.* This puts a breakpoint in the code so it will terminate cleanly once it runs.

Press Cntl-C to break out of the *Assemble* command.

> Debug returns you to the dash prompt. Now you're ready to run the code.

Type G and press *Enter*. Debug runs the code and gives you a dump of the results.

Here's what I got when I ran the code:

```
AX=2A02  BX=0000  CX=07CA  DX=0C17  SP=FFEE  BP=0000
SI=0000  DI=0000

DS=4385  ES=4385  SS=4385  CS=4385  IP=0104
NV UP EI PL NZ NA PO NC

4385:0104 CC            INT     3
```

It should come as no surprise that the output is all but inexplicable. We're running assembly-level code, after all. Here's what it all means, or at least the important parts. You can ignore all but the AX, CX, and DX registers.

```
AX=2A02
```

The first two characters, 2A, verifies that the function number you specified really did end up in the high byte of the AX register.

You can ignore the low byte (the second pair of characters) in this case.

```
CX=07CA
```

This register holds the year. Unfortunately, the number's in hex. In real-people numbers, it translates to 1994.

```
DX=0C17
```

The high byte, 0C, contains the number of the month in hex. This would be the number 12 if you had sixteen fingers.

The low byte, 17, shows the day. This is actually 23 in base 10.

Put the two together and you get the date I issued this command. December 23, 1994. Merry Christmas.

EXCEPTIONS

So far we've been talking about software interrupts. There are two other types: Hardware interrupts and Exceptions. Let's take Exceptions first because they're simpler.

If the CPU gets an instruction it cannot complete, like attempting to calculate "32/0", it would stall and the machine would crash. Not elegant. Not pretty. Not funny.

Exceptions give the machine a way out, a special set of codes to tell it how to reject the improper function and get its equilibrium back.

The CPU keeps the exception handlers in its own closely-held memory because the cause of the exception might have scrambled main memory beyond repair.

There are over a dozen exceptions. Those with numbers over 8 indicate problems in protected mode. Some examples are:

+ Exception 1 -- Divide by 0.
+ Exception 12 -- Stack Fault. The contents of a protected-mode stack became corrupted.
+ Exception 13 -- the infamous General Protection Fault. The contents of a protected-mode memory address descriptor table has become corrupted.

HARDWARE INTERRUPTS

Peripherals such as disk drives and video cards and system timers and, you guessed it, network adapters, use hardware interrupts to get the CPU's attention.

The CPU has a special pin for a hardware interrupt. Putting a voltage on that pin makes the CPU react just as if it had received a software interrupt from a running application. It looks to its registers for a function number, runs off to the Interrrupt Vector Table to find out what to do, etc.

This one pin doesn't provide nearly enough access considering the number of peripherals in the machine. So instead of sending their interrupt voltage directly to the CPU, the peripherals send their signals to an intermediate chip, the PROGRAMMABLE INTERRUPT CONTROLLER (PIC).

The PIC acts like a spider sitting at the center of radiating strands of webbing. When an unfortunate fly lands on one of the strands, the spider senses the direction and hurries out to see if there's a meal jiggling the web.

These strands are called Interrupt Request lines, or IRQs.

+ A PIC is an eight-bit device, so it can only handle eight interrupts.
+ In AT-class machines (and higher), a second PIC connects to the first like plugging one surge suppressor into another. This gives 14 possible IRQs (you lose 2 to the connection line.)

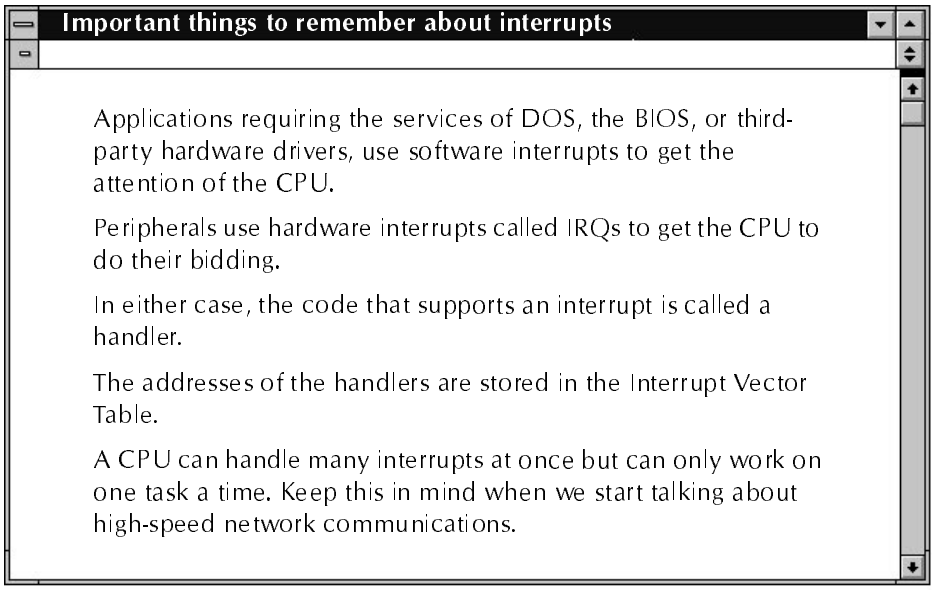
IRQs for standard devices such as disk drives and COM ports and printer ports are kept in ROM.

Add-on adapters like network cards load their drivers and get their IRQs sometime after the machine boots. They also install an interrupt handler

A peripheral or adapter lays claim to an IRQ by having the appropriate bus connector and a driver that installs a handler and rehooks the Interrupt Vector Table.

Here's how it works:

+ Let's say the network card driver has loaded and completed the preliminaries of getting an IRQ and installing an interrupt handler. Now it has a byte it wants to transfer to main memory. It puts that byte on the data bus of the card and signals the interrupt controller by putting a voltage on its IRQ line.
+ The interrupt controller senses this IRQ voltage and converts it to a hexadecimal software interrupt number that the CPU will understand by adding 8 to the IRQ number; in this way, IRQ 3 results in an INT 0Bh. The interrupt controller then taps the CPU on the shoulder saying, in effect, "I have a message from a peripheral. Do you have a moment to deal with it?"
+ If the CPU isn't busy, it fetches the INT number and the address of the interrupt handler from the PIC and intitiates the interrupt. The subsequent sequence of events matches that of the regular software interrupt we just went through.



Network Adapters

It just wouldn't be computerish if there weren't at least a half-dozen names for the devices that, along with the cables and terminators and assorted hardware, form the physical layer of the network.

NETWORK ADAPTERS – this name uses the same logic as video adapters, I suppose, where a signal gets passed to an outside device, though you don't usually hear modems called telephone adapters.

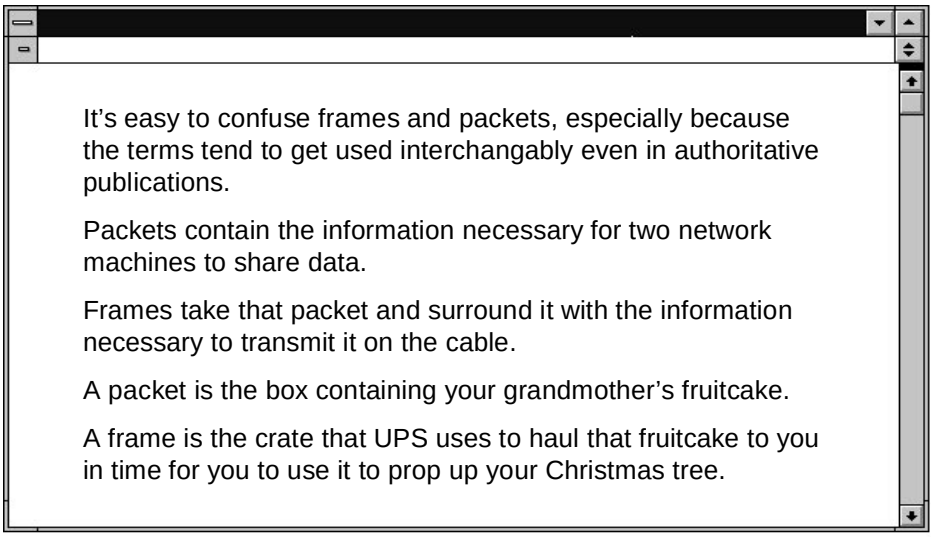
NETWORK INTERFACE CARD (NIC) -- this is a popular name. Using it with the right kind of offhanded tone makes you one of the networking cognoscenti.

BOARDS -- short for circuit boards. This is common nomenclature in Novell literature.

CARDS -- as in "Put the card in the right slot, sir." Also another way of drawing the Old Maid.

Whatever name you use, network adapters have the same function. They're radio stations. When Proctor and Gamble has soap to sell, it buys time on KNST and prays that John C. Scott won't air the spot. When an application has data to send to another machine on the network, it calls for the services of the NOS.

+ The NOS fiddles with a few production details then hands the data over to the network adapter,
+ The adapter driver and the transport driver work together to chop the data into discrete nuggets called packets. The packet contains the destination name, the source name, and information that the network operating system at the receiving machine will use to interpret the data.
+ Once the packet is built, the network controller chip on the adapter takes over. This chip takes the packet and attaches the ethernet node ID of the destination, the ethernet node ID of the source, an inventory (called a cyclic redundancy check, or CRC), and perhaps a tick mark to help with routing. This encapsulated packet is called a *frame*.



It's easy to confuse frames and packets, especially because the terms tend to get used interchangably even in authoritative publications.

Packets contain the information necessary for two network machines to share data.

Frames take that packet and surround it with the information necessary to transmit it on the cable.

A packet is the box containing your grandmother's fruitcake.

A frame is the crate that UPS uses to haul that fruitcake to you in time for you to use it to prop up your Christmas tree.

The frame transmission method differs depending on the type of network: ethernet compared to token ring, for example. Some aspects of adapter design remain constant, though.

Each computer architecture requires a different style of adapter or an adapter with several different output ports. Adapters come in a variety of styles for the following architectures.

ISA (Industry Standard Architecture) bus cards using 8 or 16 bit data lines.

EISA (Extended ISA) bus cards using 32 bit data lines and a variety of nifty performance enhancements offered by the EISA architecture.

PCI (Peripheral Control Interface) bus cards using 32 and eventually 64 bit data lines.

Parallel-port adapters that fit on the computer's printer port so no dangerous tools like screwdrivers are required for installation.

PCMCIA (Personal Computer Memory Card International Association, now thankfully renamed to PC Card) cards that fit in special slots on laptop computers.

And now that networking has become *de rigueur* in the corporate world, many computers now come with networking chips built right into the motherboard.

Adapter manufacturers are responsible for providing drivers to make them work with various networks, but network vendors, and Artisoft is no exception, often write or commission the writing of drivers for the most popular adapters to encourage users to buy their software.

Examples Of Adapters And Their Lantastic Drivers

Vendor	Adapter	Driver
Microdyne	NE2000	NEX000.EXE
Standard Microsystems	WD8003	WD8003.EXE
3COM	3C503	3C503.EXE
Thomas Conrad	Arcnet	TCARC.EXE
IBM	Token Ring	IBMTOK.EXE
XIRCOM	Parallel Port	PE3.COM

ADAPTER OPERATION

Adapter drivers work hand-in-hand with the transport driver to ferry data back and forth to and from the cable. In the case of Noderunner SI adapters in a LANtastic network, the two drivers would be NR.EXE and AILANBIO.EXE.

NR and AILANBIO become aware of each other when they first load via the general purpose multiplex interrupt, 2F. As the term applies in this instance, multiplexing permits two drivers to share a common interrupt. They are like lost siblings, each treasuring one half of a token. When introduced, they find that the two halves match and they immediately become inseparable.

Here's how the MPX trick works:

+ When NR loads, it uses INT 2F to check for interlopers on its default MPX handle of C7.

  If some other driver has preempted C7, the driver will give an error. An alternate MPX number between C0 and FF can be specified with the MPX switch on the command line.

  Example: `NR  MPX=C8`

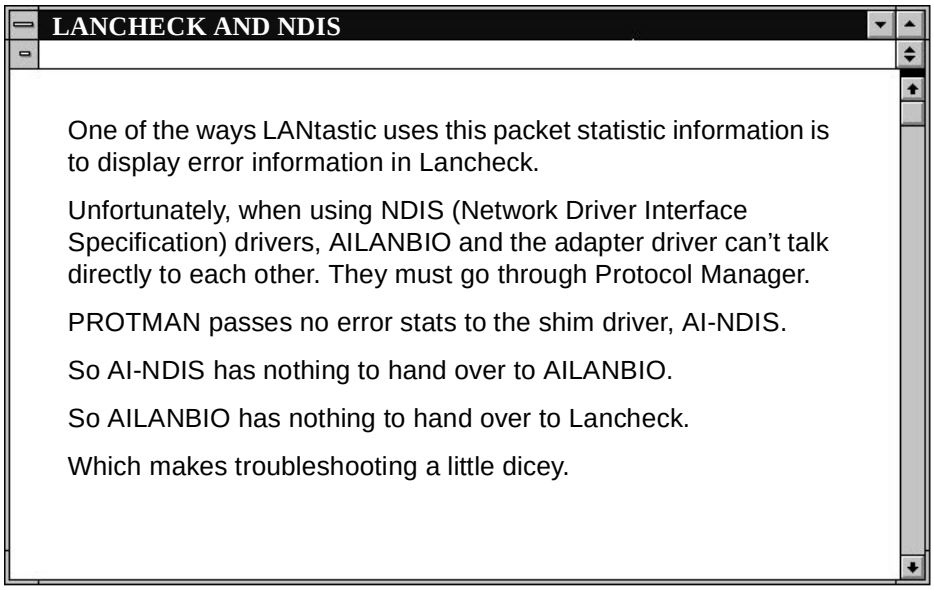
  If the MPX handle is available, NR will take over the lower byte of the vector.

+ When AILANBIO loads, it also uses INT 2F with the same MPX number looking for NR. If the driver's there, AILANBIO binds to it and takes the upper byte of the vector.

  If it finds no driver, AILANBIO coughs up an error message and refuses to load.

+ Having done its work, the MPX interrupt is not used again.

Having found each other, the two drivers exchange tidbits about each other. NR will store statistics about packet communication where AILANBIO can find it. For its part, AILANBIO learns what IOBASE the driver's using and where to store data prior to processing by the network controller chip, or NIC controller.



OPERATION OF THE ADAPTER DRIVER

The NIC controller performs the grunt labor of the data transfer. It scoops up a shovelful of data, boxes it up into a packet, and transmits the packet out onto the wire. Once it's satisfied that the packet got delivered without collision, it goes back for another shovelful.

The adapter driver determines the size of the shovel via the command-line switch PACKET_SIZE.

The XEROX rules for Ethernet limit the packet size to 1500 bytes, but this isn't a hard-and-fast standard. LANtastic networks can send and receive packets up to 4300 bytes.

When communicating between different networks, though, it might be necessary to change the packet size to accomodate the other network's limitation.

The value for PACKET_SIZE can be set between 600 and 4300 bytes.

Example: `NR PACKET_SIZE=2200`

Check the operation of the switch as follows:

Load NR with a PACKET_SIZE of 3000 and tack on the VERBOSE switch.

```
NR PACKET_SIZE=3000 VERBOSE
```

In the verbose readout, note that *Network Packet Size* is 3000.

Also note that *Packet Type* is XEROX.

Before NOS 5.0, LANtastic adapter drivers defaulted to IEEE 802.3 packets.

Beginning with NOS 5.0, the drivers default to XEROX (or XNS, for *XEROX Network Services*) packets. Novell calls this an ETHERNET II packet. You don't expect them to plug the competition, do you?

FURTHER EXPLANATION OF XEROX AND IEEE PACKETS

XEROX and IEEE packets have similar construction.

- 8 byte preamble
- 6 byte destination
- 6 byte source
- 2 byte [*]
- 46-1500 bytes (data)
- 4 byte CRC

The only significant difference is that 2-byte [*] field.

In XEROX packets, this field contains the network TYPE, because many different vendors use Ethernet with proprietary transport protocols.

In IEEE packets, the 2-byte field contains the LENGTH of the data field.

The IEEE or XEROX switch only affect the transmission. AILANBIO works with either packet type it receives.

Load AILANBIO with the VERBOSE switch.

```
AILANBIO VERBOSE
```

Note that it used the same MPX number as NR.

Run LANCHECK.

You will get one line representing the local adapter.

Press *Enter* on that line. A DETAILED ADAPTER INFORMATION screen appears.

Find *Max Packet Size*. Note that the value is 37 bytes less than the size set by the PACKET_SIZE switch. These bytes are "hidden" from Lancheck because they're used by the program to gather its information about the frames.

Escape out of LANCHECK back to the command prompt.

Refer to module HDW02, Fundamentals of Ethernet Communications, for a detailed description of the network adapter operation.

Determining Proper Packet Size

The ideal PACKET_SIZE setting involves a tradeoff between throughput for the individual adapter and network efficiency.

The express lane at a grocery store can handle a high volume of traffic, but each customer can't carry many groceries (unless it's that grouchy granny with the full basket that always seems to be in front of me whenever I shop.) The standard lanes move fewer customers but each one can pile items on the conveyor belt until the mechanism breaks.

When an adapter receives a packet, if one bit, just one, has changed or gotten trashed, the CRC will fail and the whole packet must be shipped out again.

On a clean wire in a small network, a large packet stands a good chance of getting to its destination unscathed.

But as traffic increases, or if noise or bad connections choke the wire, smaller packets stand a better chance of avoiding collisions or corruption. This reduces the amount of data that can be passed in one stroke but reduces the number of retransmissions.

Also, it doesn't do any good to increase PACKET_SIZE on the adapter driver without increasing the buffer sizes in AILANBIO and REDIR and SERVER. We'll discuss this in detail a little later.

In a LANtastic network, if AILANBIO detects a discrepancy between the size of the incoming packet and the size of the adapter driver's buffer, it will send a command to the sending machine to reduce the packet size.

Misc Adapter Driver Features

There are a couple of other bits of information in the verbose listing of NR that warrant interest.

+ NETWORK BUFFER SIZE -- displays the memory available on the adapter. Noderunners have a 32K SRAM chip. NR2000A adapters have a socket for a second 32K SRAM, for a total of 64K of on-board memory.

  AEX cards have two 8K SRAM chips that can be replaced with two 32K chips, also making a total of 64K.

  Adding memory to an adapter can only improve performance if limited card storage caused a traffic bottleneck.

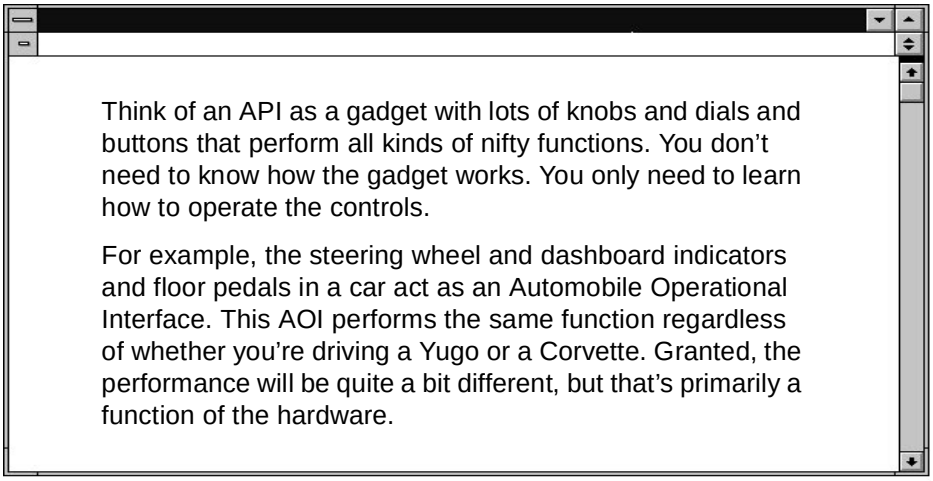
  You can check if this is happening by running Lancheck after the network's been up for a while. If the *Resource Exhaust* field contains any tallies, then the extra memory will help.

+ TRANSMIT BUFFERS -- affect how the on-board memory gets divvied up. Normally this memory splits half and half between transmit and receive buffers. Increasing Transmit Buffers will steal memory from the receive buffers. This could conceivably help a workstation, but it invites problems because a small number of receive buffers will cause resource exhausts.

Contrary to what you might think, NetBIOS is not a slavering beast lurking in the dark, waiting to pounce on unsuspecting techs. If it is a beast of any kind, it's more like a puppy that's tough to train but eager please.

Once again, NetBIOS stands for Network Basic Input/Output System. It isn't a standard like RS-232 or CCITT V.25. It's more of a gentleman's agreement, like the Hayes AT command set. Sytek, Inc., developed it for IBM to support their broadband network and everyone else pretty much followed the general style.

NetBIOS is an *Application Programming Interface (API*).



Think of an API as a gadget with lots of knobs and dials and buttons that perform all kinds of nifty functions. You don't need to know how the gadget works. You only need to learn how to operate the controls.

For example, the steering wheel and dashboard indicators and floor pedals in a car act as an Automobile Operational Interface. This AOI performs the same function regardless of whether you're driving a Yugo or a Corvette. Granted, the performance will be quite a bit different, but that's primarily a function of the hardware.

NetBIOS is commonly associated with ethernet networks, but that's only because they became popular about the same time and grew up together. NetBIOS works just as well on token ring networks, ArcNet networks, and mammoth campus-wide Fiber Optic (FDDI) networks.

NetBIOS doesn't actually do any work, not if you think of work as moving data from one place to another on the network. The network adapter card takes care of that. NetBIOS performs the same function as a host at a masked ball:

+ Identifies each machine in the network by a unique name and keeps a table of those names along with the machine's true identity (a unique number burned into the ROM of the network adapter.)
+ Establishes and manages permanent connections between pairs of machines. These connections are called *sessions*. A session consists of a *receiver* and a *sender* or, if no data is actively changing hands, a *listener* and a *caller*.
+ Packages data for transmission by the network interface card.
+ Acknowledges when data has been received intact.

Individual NOS vendors develop their own code for implementing the NetBIOS interface and package it as either a device driver or an executable. Artisoft's LANtastic NOS uses AILANBIO.EXE (Adapter Independent LAN BIOS)

.

EXAMPLES OF THIRD PARTY NETWORKS AND TRANSPORT DRIVE

NOS	NetBIOS driver
Windows for Workgroups and NT	NETBEUI (NetBIOS Extended User Interface)
IBM PC LAN Support	DXMT0MOD.SYS
OS/2 LAN Server	NETBIOS.OS2
Artisoft LANtastic for OS/2	AILANBIO.OS2
Novell Netware	NETBIOS.EXE (a NetBIOS emulator)

Generally, different flavors of NetBIOS are not compatible. An adapter running LANtastic's AILANBIO will not be able to talk directly to an adapter running Microsoft's NETBEUI.

These are the days of increasing customer demand for connectivity, however. NOS vendors are scrambling to make their NetBIOS implementations transparent by loading them in stacks on the same adapter. Two of the more popular methods are

+ Microsoft's Network Device Interface Specification (NDIS)
+ Novell' Open Device Interface (ODI) are two examples.

NetBIOS, as a gadget, only manages the data delivery between computers. it's up to the application using its services to use that data successfully. This is analogous to the way DOS works. An application must use DOS to get access to a file, but from then on the application reels in the data according to its need.

Applications that make direct use of NetBIOS services are said to be *NetBIOS-aware*. The network Redirector and Server programs, REDIR.EXE and SERVER.EXE, are two such programs.

EXAMPLES OF NETBIOS-AWARE PROGRAMS

Winfax Pro from Delrina, a network fax-sharing program

Articom, a network modem-sharing program

Carbon Copy from Microcom, a network console-sharing program

Watcom's SQL, a network database-sharing program

Artisoft's Linkbook, a network Clipboard-sharing program

Wordperfect Office Mail, a network email program

Preparing the Network Message

A network, whether it's LANtastic or some other less attractive brand, works something like Federal Express.

+ You package up your shipment according to their rules.
+ You give it to their agent.
+ You pay your tab.
+ Federal Express takes care of delivery. You don't care how they do it. You just want delivery to occur as soon as possible.
+ You wait for a phone call from the recipient thanking you profusely for your thoughtful gift (unless you're the parent of teenagers, in which case you'll have to do the calling.)

One rule both of Federal Express and NetBIOS is that every package must have a shipping manifest that lists exactly what's being sent, who sent it, where it's going, who's supposed to receive it, and what to do in case of trouble.

SPECIAL WARNING

To make a clear presentation of the way NetBIOS works, you'll have to look at a line or two of C code.

DON'T BE CONCERNED.

The Surgeon General has determined that occasional exposure to programming is not considered a serious health risk.

At worst, you might suffer a slight concussion if you nod off and hit your head on the table.

The shipping manifest used by NetBIOS is called a *Network Control Block* (NCB). The Network Control Block takes the form of a *data structure*.

A data structure is a block of information consisting of separate fields stored together as a single entity in the computer's memory or on disk.

Example of data structures

A contact management program might contain a database with the following information for each customer:

```
NAME:[_____________________________]
STREET:[_____________________________]
CITY:[_____________]
STATE:[__]
ZIP:[_____]
AGE:[___]
CREDIT LIMIT:[$_______.__]
```

To store this information, the program builds a data structure called CUSTOMER and uses it as a template for storing the required information, just like filling in a preprinted rolodex card.

```
#define struct CUSTOMER
char      NAME[40]
char      STREET[40]
char      CITY[20]
char      STATE[2]
char      ZIP[5]
int       AGE
float     CREDIT_LIMIT
```

Note that, because we're using a computer and not a rolodex, the data structure must also include the type of data (character, integer, floating point decimal) that each field will accept. Computers are nothing if not fussy.

Network Control Block Structure

The next three pages contain the fields in a Network Control Block data structure. Remember, the application, not NetBIOS, constructs the NCB and fills in the applicable fields. The Federal Express agent will not fill in the shipping manifest for you. You're lucky if he doesn't drop your package.

Don't confuse this NCB structure with the frame or packet that will be transmitted. NCBs don't control the form of the frame. That's determined by the adapter driver and the NIC controller.

Field Name	Description
NCB_COMMAND	Command Name This is a NetBIOS command, one of the 21 it keeps in its tool kit. Those commands will be listed a little later in the module. This field is only 1 byte wide, so instead of using the full command name, the application inserts a number corresponding to the command. For example, the CALL command is number 0x10h.
NCB_RETCODE	Return Code When an application issues a NetBIOS command, NetBIOS always gives back a Return Code. Under most circumstances, this number will be 00, indicating that all went well. If an error occurs, then NetBIOS returns a number corresponding to the error. NOTE: This is not the same as a NetBIOS acknowlegement. ACKs, as they're called, happen at a lower level.
NCB_LSN	Local Session Number Two machines that want to exchange data must first establish a *session* between each other using NetBIOS. This session gets a Local Session Number (LSN) that the application uses in much the same way as a file handle. Communications between the two machines reference this LSN rather than going through the tedious drill of filling in the name of the sender and receiver each time.
NCB_NUM	NCB Number NetBIOS keeps a table of all the names that are using an adapter. When it adds a name to this table, it assigns a number to the name. It can't help doing this. It is, after all, a computer program. The application building this NCB (such as REDIR) uses this number rather than the name to improve efficiency and limit the size of the NCB.
NCB_BUFFER	Message Buffer Pointer NetBIOS applications are like Washington journalists or soap opera characters. They spend their lives transferring tidbits of information back and forth to each other. This might be a NOS command like "Report Printer Queue Contents" or perhaps a batch of records to store in a database or it could be a stream of bits headed for a printer or modem. In any case, as part of the preparations an application makes before calling NetBIOS, it puts the data in a buffer of its own making. This is not a NetBIOS buffer. In the case of REDIR, it would be a Redirector buffer. In the case of Articom's A-SERVER, it would be an A-SERVER buffer. It would be slow and wasteful to copy the contents of this buffer into the NCB. Instead, the application passes NetBIOS the address of the buffer in offset:segment format (eg, 0434:C400.) When NetBIOS is ready to process the data down to the adapter card, it jumps to this address to get the contents of the buffer.
NCB_LENGTH	Message Buffer Size in bytes NetBIOS limits the amount of data that can be sent with one NCB to 64K. The application might have megabytes to send across the network, but it has to do so in 64K chunks. Here's a quick explanation, if you're not intimidated by hex arithmetic. This field is 2 bytes wide. The largest hex number that a 2-byte field can hold is FFFFh. That translates to 65535d, or 64k. Think of the application's message buffer as a holding pen. The old cowpoke has ten-thousand head of steer to move onto the train to Wichita. It only makes sense to herd them up into boxcar-sized groups. If the command in the NCB_COMMAND is CHAIN_SEND (as opposed to plain old SEND), then NetBIOS will link an additional 64k buffer to this NCB. See the next field for a description of how it accomplishes this trick.

NCB_CALLNAME	NetBIOS Name of the Remote Computer This 16-byte field has enough room for the standard 15-character NetBIOS name plus one extra byte for C's hidden ASCIIZ string terminator. The application uses this field to tell NetBIOS (and eventually the adapter driver) who to put on the packet's destination address. If two machines have already established a session with each other, recall that they use the LSN (Local Session Number) rather than their names. Therefore, this field is superfluous and can be used for other purposes. In addition to the original NetBIOS non-standard, IBM implemented a new command, the CHAIN SEND command. If the application is using CHAIN SEND, then it fills in this field with a pointer to a second data buffer and the size of that buffer (64K maximum.) Doubling the maximum amount of data that can be sent with one NCB is handy, but the real power of CHAIN SEND is the way it facilitates network operations. LANtastic uses a set of data structures called SNBs to tell REDIR and SERVER how to handle incoming and outgoing data. These are compact, generally less than 150 bytes. CHAIN SEND permits the SNB and the data associated with it to be sent with the same NCB.
NCB_NAME	The NetBIOS name of the local adapter This could be the 6-byte ethernet node ID from the adapter's ROM. The NOS, though, can fill in this field with one of the names it has assigned to the adapter.
NCB_RTO	Receive time-out Once two machines have established a session, the application issues SEND and RECEIVE commands to pass data. This field tells NetBIOS how long to wait for data after a RECEIVE command has been issued.
NCB_STO	Send time-out The time NetBIOS will wait after a SEND command for the receiver to acknowledge. If it times out, the session will be canceled.
NCB_POST@	Pointer to POST routine (offset:segment) The application calling NetBIOS should have a plan for what to do when the command issued by this either completes or errors out. The code that implements this plan is called a *post handler*. The application places the address of this handler in this field.
NCB_LANA_NUM	LAN Adapter Number This field is not nearly as auspicious as it sounds. It merely indicates which of two possible adapters will get the NCB command. Hex 00 indicates primary adapter. Hex 01 indicates alternate adapter.
NCB_CPLT	Command Complete Status NetBIOS sends back a code after it gets done processing the command in the NCB_COMMAND field. It puts that code here in this field. Immediately after receiving the command, NetBIOS puts an FFh in this field. Then, once it completes the command, it replaces the FFh with the final return code. The application must check this field periodically to get the return code when it finally arrives. If the return code is 00, the command executed without error. Return codes of something other than zero indicate an error. See the error list further on in this module.
NCB_RESERVE	Reserved for NetBIOS

USING NCBS

Remember the children's game *Mousetrap*. In it, the players spend hours building an elaborate contraption. The winner gets to push a little steel ball that sets off a symphony of swiveling, bouncing, slinging, ratcheting parts that eventually drop a plastic cage down on a plastic mouse. This happens so quickly, though, that the finale comes as something of an anticlimax.

The same is true of NetBIOS commands.

+ The application spends time building an NCB and painstakingly filling in all the required blanks.
+ It puts a buffer of data in memory and points at it with an address pointer.
+ It figures out which POST handler to use to handle the return message and puts the address of that handler in the NCB.
+ If it's going to receive some data as a result of issuing the NCB, it sets up a data structure to store the data.
+ Finally, it puts the address of the NCB into the appropriate registers of the CPU and delivers the whole shebang to NetBIOS for processing.

The push that gets this electronic mousetrap sputtering along is, as you might expect, an interrupt; in this case, INT 5C.

Once the NetBIOS application issues the INT 5C, things happen fast.

+ *Rattle*. The CPU puts down its knitting and checks the Interrupt Vector Table at location 5C.
+ *Clunk*. It finds the the address of the interrupt handler placed there by NetBIOS.
+ *Pitter-Patter*. The CPU jumps to that address and finds code that tells it what to do with the NCB based on the contents of the NCB_COMMAND field.
+ *Chunka-Chunka*. NetBIOS chops up the message into chunks based on the frame size it's been programmed to use.
+ *Ptui*. The message gets passed on to the adapter driver.
+ *Sigh*. The CPU returns to its knitting until the adapter driver starts bothering it with hardware interrupts for the data transfer.

NetBIOS can't relax yet, though. It has initiated the command, but it can't sleep soundly until it knows the command successfully completed.

The copy of NetBIOS running on the receiving machine will send back a message saying, "Safe and sound." or "So sorry, try again."

There are two ways NetBIOS can pass the time waiting for this message: *wait* and *no wait*.

+ WAIT tells NetBIOS to block the execution of other programs and sit tight until it receives the acknowledgement. A successful transfer returns a code of 00 to the NCB_CMD_CPLT field. An unsuccessful transfer returns an error code. Error codes are listed later in the module.
+ NO WAIT releases the CPU to do other tasks. The application then must either continually check the status of the NCB_CMD_CPLT field like an anxious cook hovering over a pot of fudge. Or it can put a pointer to the next batch of code in the NCB_POST field and wait to be notified of the results.

A "1" in the high bit of the NCB_COMMAND field tells NetBIOS to use the *no wait* option. For the hexadecimally inclined, this is the same as adding 8 to the *wait* version of the command. For example, the ADD_NAME_WAIT command has the number 0x30h while the sister command, ADD_NAME, has the number 0xB0h.

The NO WAIT option is preferable from a user's perspective, but in a multitasking environment, where timing can be critical, NO WAIT programming can bollix up other processes running on the machine.

The application making the NetBIOS call determines which option to use.

Follow the Bouncing Packets

In an actual network, the volley of NetBIOS commands necessary for exchanging a buffer of data includes a few more commands and acknowlegements, but overall the process remains amazingly simple. The gadget called NetBIOS isn't all that complicated. DOS contains hundreds of function calls. NetBIOS contains only 21. The inevitable list is on the next page, but before you read it lets trace a sample NetBIOS exchange and introduce the important commands.

ADDING NAMES

One of the primary responsibilities God gave to Adam (and, once supposes, Eve) was to name the plants and the beasts of the world. Proper names, then, carry something of a historical imperative to which NetBIOS is not immune.

Every adapter on the network must have a unique name. The error, "Name Already In Use," displays if the name is not unique.

NetBIOS keeps a table of names for the adapters in the machine. The NAMES switch on the AILANBIO line determines the size of this name table.

The NetBIOS command ADD NAME puts new names in this table. Before a name is added, NetBIOS polls the network to see if it's in use anywhere.

A similar command, ADD GROUP NAME, allows the adapter to become one of a group of adapters with the same name. In this way, one adapter can send the same packet to a designated group, thereby minimizing network traffic.

For example, a fax-sharing NetBIOS application could assign a group name of FAXUSER to each workstation that has the client program in memory. A machine called FAXSERVER then would only need to send out one datagram with a destination of FAXUSER to communicate to all the workstations.

Individual and group names are both kept on the same name table. The group name can exist on several adapters, obviously, but must be unique on any given name table.

An adapter can carry more than one name. For example, you can assign a REDIR name of RAPUNZEL and run LANCHECK with a name of PRINCE_CHARMING. Both names will be added to the name table.

You could take the t-connector off the network adapter and issue a non-unique ADD NAME command, but when you replace the T-connector and send out the first packet, you'll probably get an error, "Your name is not unique on the network." The error handling routines of the NOS will probably reset the adapter or otherwise remove the offending name from the name table.

Once the adapter has a unique name, it can join the network.

SESSIONS

NetBIOS requires that two machines enter into a relationship with each other before they can exchange information. This relationship is called a *session*.

+ A session forms a virtual circuit, a railroad track of sorts that shuttles freightloads of information back and forth between the two adapters.
+ An adapter can have more than one session but, like a rail yard roundhouse, the track can only carry one train at a time.

The number of simultaneous sessions AILANBIO can set up is set by two command-line switches, MAX_SESSIONS and SESSIONS. The default for both values is 32.

The reason for having two settings is that IBM's original NetBIOS implementation made allowance for changing number of sessions on the fly. The total sessions, however, can't exceed the maximum value designated for NetBIOS when it first loads.

The same is true for the number of Network Control Blocks, or NCBS. AILANBIO has two switches, NCBS and MAX_NCBS, and sets them both at a default of 32.

This does not apply to the NAMES switch for AILANBIO. There is no MAX_NAMES setting.

LANtastic networks (Redirector and Server, to be specific) do not dynamically change the number of NetBIOS sessions nor NCBS. The potential for saving memory isn't worth the overhead in coding.

CHANGING SWITCH VALUES

If you need to increase the number of sessions--say for example you have a server feeding more than 32 workstations-- then you must increase all five settings: SESSIONS, MAX_SESSIONS, NCBS, MAX_NCBS, and NAMES.

Even though LANtastic doesn't use them, AILANBIO keeps the separate switches for compatability with other NetBIOS applications that might use this feature.

While we're on this subject, what if we're running LANtastic on someone else's NetBIOS--we can do this if the third-party NetBIOS is compatible--and we need to increase Sessions or NCBS. An example would be some flavors of Microsoft's and IBM's NETBEUI. These third-party NetBIOS programs initialize with a small number of sessions and NCBS, generally not enough for LANtastic to run effectively.

Because Redirector and Server don't issue on-the-fly changes to sessions and NCBS, we need to instruct the BrandX NetBIOS to up the ante before loading the network.

The program that does this is NBSETUP.

You would run NBSETUP along with a command-line switch for the parameter, either NCBS or SESSIONS or both, that you want to adjust.

Example: `NBSETUP SESSIONS=32 NCBS=32`

You can also go pot limit by way of the MAX switch.

Example: `NBSETUP MAX`

This will increase NCBS and SESSIONS to the maximum supported by the BrandX NetBIOS.

MORE ABOUT SESSIONS

Back to our example. We said earlier that NetBIOS is a connection-oriented protocol that sets up virtual circuits called sessions between two communicating machines.

To establish this session, NetBIOS on one machine issues a LISTEN command.

> The LISTEN command tells the adapter to be on the alert for an incoming packet from an adapter it specifies by name. It says, in essence, "Listen for a packet from CLAUDE."

NetBIOS on the second machine issues a CALL command directed at a specific machine name. For example: "Call NANCY."

+ If no adapters on the network have the name NANCY, NetBIOS returns an error, "Cannot Locate Network Name."
+ If an adapter with the name NANCY exists on the network but does not have a pending LISTEN, NetBIOS returns an error, "Machine NANCY is not listening."
+ If an adapter with the name NANCY exists on the network and has a pending LISTEN, then the two NetBIOS commands complete successfully. NetBIOS at both ends gives the application a Logical Session Number (LSN) that it uses to simplify addressing in subsequent data exchanges.

The remainder of this page is intentionally blank. Proceed to the next page for a list of NetBIOS commands and their functions.

NetBIOS Commands

Here are the NetBIOS commands in all their glory:

ADAPTER STATUS	You're familiar with the information returned by this command. LANCHECK uses it to build its display. It returns the following information: **Card ID –** for Ethernet cards, this is the node ID number burned into the adapter's ROM **Release Level –** the version of NetBIOS running on the machine **External Option** -- Jumper configuration on the original PC Network Adapter **Adapter Type –** FE for ethernet adapters and FF for token ring adapters **Old or New Parameters** -- tells whether NetBIOS was started with old or new parameters **Reporting Period –** the amount of time that NetBIOS has been running in memory **CRC Errors** -- the number of packets that were received in a corrupted condition. CRC stands for Cyclic Redundancy Check. A CRC picks several bits in the frame and plugs the value into a rather complex polynomial. The outcome of this calculation is a 4-byte, 32-bit number. The receiving adapter performs the same calculation. If it does not arrive at the same answer, the packet is discarded. **Alignment Errors –** the preamble of an ethernet packet consists of 62 alternate 1's and 0's with a 11 at the end. The card probably won't see this entire preamble due to noise and such on the wire. (Also, the transceiver ramps up the amplitude to prevent ringing, so the initial part of the preamble is often below the grass, as they say.) If the card doesn't get enough of the preamble to be absolutely sure of the start bit, or it can't sync with the signal, it will discard the packet as an alignment error. **Collision Errors** -- the number of times a collision was detected after a packet was broadcast **Unsuccessful Transmissions –** the number of transmissions for which there was no corresponding acknowledgement from the receiver **Good Transmissions** -- the number of transmissions for which there were corresponding acknowledgements from the receiver **Good Receives** -- the number of packets received without error **Retransmissions –** the number of packets that had to be retransmitted due to some error **Exhausted Resource Count** -- the number of times that data was lost because NetBIOS ran out of buffers **Available NCBS –** the difference between the number of Network Control Blocks (NCBs) in use and the maximum number of NCBs available **Maximum Current NCBS –** set by the RESET command **Maximum Possible NCBS** -- depends on adapter type **Maximum Frame Size –** the largest frame available for transmission. This is generally 1463 bytes unless changed by a switch on AILANBIO. **Name Count** -- the number of names in the NetBIOS name table **Name Table** -- the name, number, and status (Registered or Deregistered, Unique or Group) of each name in the NetBIOS name table **Pending Sessions** -- the number of active send/receive pairs (each one constitutes a session) **Maximum Current Sessions** -- the difference between the maximum possible sessions and the active sessions **Maximum Possible Sessions** -- set by the MAX_SESSIONS switch on AILANBIO

ADD NAME and ADD GROUP NAME	Every machine running NetBIOS keeps a table of the names with whom it carries out transactions. The NAMES switch on the AILANBIO line determines the size of this name table. The first name on this table belongs to the machine itself. Before a name can be added to the list, NetBIOS first polls the network to see if the name is in use anywhere else. The error, "Name Already In Use," displays if the name is not unique. A machine can have more than one name. This is how REDIR and NET USER can keep separate names. In addition to establishing sessions between individual machines, NetBIOS also will let one machine communicate with many machines. Individual and group names are both kept on the same NetBIOS name table. Group names can be in more than one name table at the same time but must be unique on the network. You can take the t-connector off the network adapter and issue an ADD NAME with a name that is used elsewhere in the network. As soon as you replace the T-connector and send out the first packet, however, you'll get the error, "Your name is not unique on the network." The error handling routines of the NOS might reset your adapter or otherwise remove that name from the name table. The Add Name command only affects the name table on the machine that issues it.
CANCEL	This command terminates a specified NCB. For example, say you have a pending LISTEN that you want to cancel. You would issue an NCB for the CANCEL command and point the message buffer at the address of the NCB for the LISTEN command. This snuffs the LISTEN. Did you ever see the movie *Zot.* Tom Posten played a university professor who discovered an Egyptian relic in his back yard. Painted on the relic was an ancient incantation that bestowed upon the sayer the power to kill merely by pointing at someone and saying the secret word, Zot. Think of CANCEL as the Zot of NetBIOS. A NetBIOS application (and remember, Redirector and Server are NetBIOS apps) must be careful about issuing a Zot, I mean a CANCEL. It could play havoc with the system if the canceled command leaves a DOS file open or a Windows message hanging in the queue.
CALL and LISTEN	These two commands work in conjunction to establish a session. NetBIOS on one machine issues a LISTEN. The NCB remains active, patiently waiting for a call like a lovestruck teenager. Once it gets the CALL (with due apologies to my Baptist friends,) NetBIOS on both machines establishes a *session*. The end result is a Logical Session Number (LSN) at each end that the application can use when issuing SEND and RECEIVE commands.
SEND and RECEIVE	Once two machines have a session going between each other, they can alternately SEND and RECEIVE data just like coworkers with walkie-talkies. "D'ya want lunch? Over." "Roger. Over." "My name is Herbert. Over." "You stole that joke from *Airplane*. Over." With computers, it's a little more complicated. Machine One issues a LISTEN. Machine Two issues a SEND. Machine One receives the message and returns an acknowledgement. At that point, both NCBs end successfully. Machine One must issue another LISTEN before Machine Two can issue another SEND. One of the responsibilities of the NOS is to compare the total size of a given transaction (for example, a file transfer) with the amount of data that has already arrived. As long as there's more to come, it keeps issuing RECEIVE commands so that it will be ready for the next SEND.

RECEIVE ANY	A special version of RECEIVE that instructs an adapter to be ready for a SEND from any of its active sessions. By contrast, a normal RECEIVE must specify a particular Local Session Number (LSN.)
CHAIN SEND	A special version of SEND where a second buffer of data is attached to the first buffer. AILANBIO uses chain sends by default unless told not to via the NO_CHAIN switch. Don't confuse this command with the operation of the adapter card. The two entities being chained with CHAIN SEND are buffers filled with application information, not packets travelling along the wire.
DELETE NAME	Removes a name from the name table. One of the functions of NET LOGOUT is to remove the remote machine's name from the local name table. The remote machine will also issue a DELETE NAME. "We're through, Mary. I'm taking your name out of my rolodex." "Have it your way, Murray. I'm scratching your name off my list, too." "Fine. Just be that way." "Fine."
HANG UP	Terminates the session and erases the Logical Session Number It's important that the NetBIOS application close all pending commands before issuing the HANG UP. NetBIOS will complete certain critical commands before terminating the session, but it's still not polite to remove the tablecloth before the guests have finished eating.
SEND and RECEIVE DATAGRAM --- SEND and RECEIVE BROADCAST DATAGRAM	Under normal circumstances, NetBIOS requires that two machines establish a session prior to exchanging data. This *connection-oriented* design helps ensure that all data reaches its intended target. Datagrams, on the other hand, carry data from one machine to another without the formalities of a session. There is no guarantee a datagram will arrive, however, and their use in a NOS like LANtastic is limited. Broadcast datagrams are the mass mail of NetBIOS, complete with CARRIER ROUTE SORT stamped on the envelope and an address of OCCUPANT. Broadcast datagrams are used for such general purpose transactions as hailing down a server prior to establishing a session, NET PING, and server ID broadcasts. Broadcast Datagrams in LANtastic take the form of *multicast* packets. Multicast packets contain all F's in the destination field of the packet. When a server gets a multicast packet, it must look at the data field to see if its NetBIOS name (as opposed to its ethernet ID) is specified in the SNB data structure. If it is the intended recipient, it continues processing the packet and performs whatever NOS command is in the data field. If it is not the intended recipient, it ignores the packet and goes back to doing whatever it is that servers do when they aren't dealing with incoming messages. Some routers interpret a multicast packet as a strictly local entity and will not send it out across a WAN connection. This would prevent a workstation from finding and logging into a remote server. In this case, use the NO_MULTICAST switch on AILANBIO. This will convert the destination address in the packet from a multicast to a broadcast format. A router should pass all broadcast packets.
RESET	This command is the equivalent of cleaning off the card table after a long night of poker. The command: Deletes all names from the local name table except for the permanent ethernet node ID of the local adapter. Terminates all active sessions. Tosses out all outstanding NCBs. The reset command can also be used to specify a new number for the maximum number of NCBs.

SESSION STATUS	Each adapter can coordinate a number of sessions. The total is set by the MAX_SESSIONS switch on AILANBIO. A session can have one of the following status condition: Session Established Pending Listen Pending Call Pending Hang Up Hang Up Complete Abnormal End The SESSION STATUS command will report back this information for each session on the adapter.
UNLINK	A diskless workstation uses this command to disconnect itself from the server. The remote boot process establishes a special session with the server. This command terminates that session.

EXAMPLE OF NETWORK COMMUNICATION USING AILANBIO

Now that you've memorized these NetBIOS commands, let's ignore them for a moment and look at a specific example using AILANBIO.

Let's say that Redirector has a full message buffer, 64K worth of data to transfer.

Redirector builds an NCB and calls for the services of AILANBIO via INT 5C.

Recall that the adapter can transmit a maximum of 4.3K at one shot.

Once awakened, AILANBIO works cheek-by-jowl with NR to chop the data into frames based on the setting of the PACKET_SIZE switch.

Recall that the terms *frame* and *packet* have different meanings.

+ A FRAME contains data and perhaps some routing information.
+ A PACKET contains addressing information and CRC results.

AILANBIO numbers the frames sequentially so that the receiving adapter can determine if any data gets lost or scrambled. If that happens, the receiving machine will request that the frame be retransmitted. (Remember that field in Lancheck?)

Once a frame is built, the network controller chip (ALICE on the Noderunner) does the following:

+ Attaches a preamble and the appropriate node addresses at the beginning of the frame.
+ Performs a CRC check and tacks the result at the end of the frame.
+ Sends the whole sheebang through a Manchester Encoder which converts it from a Non-Return to Zero (NRZ) signal to one with integrated timing characteristics and sufficient amplitude for long-distance transmission. (See HDW02, Fundamentals of Ethernet Communication, for more details.)
+ Listens for a break in traffic then blasts the signal out onto the cable.

If a collision occurs, or the packet fails CRC at the receiving adapter, or the Manchester decoder rejects the contents, or there wasn't enough of a preamble for the on-board clock to sync with the signal, the receiving adapter will request that the sender retransmit the packet.

ERROR STATISTICS

The adapter driver compiles statistics on all the packets it handles. It stores this information in a data structure built by AILANBIO for the purpose of satisfying the ADAPTER STATUS command.

As each packet arrives at the receiving adapter, AILANBIO sends back an acknowledgement; that is, unless the sender used the SEND_NO_ACK command. As you might expect, this does not require an acknowlegement. It operates, therefore, a little like a datagram. The onus in on the receiver to keep track of the incoming frames and request a resend if one comes out of order.

ANOTHER CONFUSING POINT

The adapter has some on-board memory to hold the frames as they're processed through the controller chip. If a packet arrives while the adapter is busy, the driver temporarily stores the packet in this on-board memory.

What do you do if you're trying to get a haircut at the mall on a rainy Saturday afternoon and all the barbers are busy and all the chairs in the alcove are full? You go back to browsing and come back again later.

If there's lots of traffic directed at an adapter, or if the traffic's normal but the adapter is slow, then a packet might arrive and not have a place to wait.

When that happens, the packet is not accepted and must be retransmitted.

The adapter will tally this as a resource exhaust for potential display by Lancheck.

MORE ABOUT BUFFERS

AILANBIO also has a buffer to store frames coming or going from the adapter.

The buffer size is set by the SIZE= switch. The default size is 570 bytes. The maximum size is 4300 bytes.

The SIZE switch setting should match the PACKET_SIZE switch on the adapter driver line.

The default sizes for these two switches come close to meeting this requirement: 600 bytes for NR and 570 bytes for AILANBIO.

AILANBIO reacts differently than NR to incoming frames when its buffer is full.

It gives priority to the incoming frame. If there were already a frame in the buffer waiting for Redirector or NR, it gets the boot and the second packet takes its place. This is likely to cause a discrepancy in the frame count, forcing the lost data to be retransmitted. This lowers performance on the slow machine and increases network traffic, which hurts overall performance.

If you have a busy node, adding more AILANBIO buffers can help performance.

Why not, then, load up on buffers by default? Why not put in four buffers with 4300 bytes apiece and jack up the adapter packet size to 4300 bytes and let that sucker snarf the network for all its worth?

+ First, buffers take memory. Adding 18K of buffering to a 20K AILANBIO might make it impossible to get the entire network into upper memory.
+ Second, a node on a small network doesn't need all that buffering. Don't put bicycles into boxcars.
+ Third, you give AILANBIO more to do when you give it more buffers. Why bother papering and dusting a pantry shelf that's likely to remain empty?
+ Finally, the AILANBIO buffer on a single node is just one piece of a long string of buffers that act as surge tanks for the network. In module NOS12, Improving Network Performance, we'll take a quantitative look at how those buffers work together.

Beyond the Black Box -- Network Messaging

Now that we know how NetBIOS works, we can safely put it aside. From this point on, we'll treat NetBIOS as a delivery gadget and move to a higher level: the network operating system itself.

DOS uses a myriad of commands to control files and printers on a stand-alone machine. These commands work directly at the hardware level, like a mother guiding her child's hand when baking cookies. Redirector translates these commands into a form that Server can understand.

SERVER NETWORK BLOCK (SNB)

The command language used in LANtastic networks is the Server Network Block (SNB) protocol.

> The SNB protocol is proprietary to Artisoft. For this reason, workstations using other network operating systems cannot directly access resources on a LANtastic server.

Don't confuse the messaging protocol with the transport protocol. A packet from, say, a Novell server can be received and processed by AILANBIO on a LANtastic workstation. But unless Redirector has been set up to receive Novell commands, called Network Control Protocol (NCP), then the contents of the data will be ignored.

WHY A NOS?

We've already seen that it's possible to code an application to make direct network calls. These programs don't need a NOS, thank you very much. They can handle their own communications.

For the most part, though, NetBIOS-aware programs like these operate in specific niches.

+ A fax-share program sends faxes across the network cable.
+ An SQL program passes database information across the network cable.
+ A modem-sharing program passes data from serial ports to communication applications.

These programs are the specialty shops in the mall. Network Operating Systems, on the other hand, are the large, general-purpose retailers. They anchor the mall and provide many of the same products and services as the little shops.

A NOS makes it possible for ordinary applications to get access to the network. The app makes a call to DOS just as it normally would. The NOS catches the call as slick as a federal marshal nabbing a crack dealer and delivers it to the appropriate server.

RESOURCE SHARING

Resources offered by servers fall into two general categories: file storage of one form or another and printers of all kinds. A specilized form of file sharing is application sharing, where the executable loads and runs across the network.

A user who wants access to a server resource issues a command that tells Redirector to map a logical drive or printer device to the resource. Examples:

```
NET USE E: \\SERVER_NAME\C-DRIVE
NET USE LPT3: \\SERVER_NAME\@PRINTER
NET USE COM2: \\SERVER_NAME\@LASER
```

Once that redirection is in place, the applications on the workstation can access the information on the server just as if it were stored on a local drive or print to a port just as if the printer were cabled directly to the machine.

Tracing LANtastic Network Commands

Let's follow how this works. Remember, although NetBIOS sits in the background and facilitates this exchange, we're only discussing the upper-level network drivers now.

For the sake of example, we'll imagine that we have a two-node network with a server and a workstation.

+ The server's name is Batman.
+ The workstation's name is Robin.
+ Server Batman has only one resource in its utility belt, C-DRIVE.
+ Workstation Robin is running a DOS word processing application. We'll say it's WordPerfect.

The stage is set. Now let's watch the actors at work. As a server, BATMAN has put up a pending LISTEN for anyone who wants to log in.

The user at ROBIN issues a NET LOGIN to server BATMAN. REDIR at ROBIN performs the following:

+ Builds an SNB with the appropriate login information.
+ Builds an NCB for a CALL command that will send the SNB.
+ Issues an INT 5C to get the attention of NetBIOS.
+ Waits for the successful completion of the login.

At this point, NetBIOS doesn't know the ethernet node ID of server BATMAN. Therefore, it sends out a multicast datagram to every adapter on the network.

See the next page for an explanation of multicast datagrams.

Also, see training module NOS04, *Special Network Configurations*, for more information about datagrams and routing.

FURTHER EXPLANATION

Multicast packets have all F's in the destination field. This is a clue to the servers that this packet must be examined all the way into the data field.

Multicast packets can be processed more efficiently. Also, routers will confine multicast traffic to the originating LAN, which will reduce WAN transmissions.

However, if the server is on the other side of the router, the multicast packet won't find it. In this case, the NO_MULTICAST switch must be added to the AILANBIO line.

This forces the adapter to send out broadcast packets, which increases traffic through the router but makes sure the datagrams get through.

When the adapter on server BATMAN receives the multicast packet, AILANBIO examines the data field and finds its own name.

BATMAN then responds to the CALL as follows:

+ NetBIOS sets up a session.
+ NetBIOS gives the session an LSN (Local Session Number) that will be used for future transactions.
+ SERVER checks access rights and adds ROBIN to its current user list.
+ SERVER builds an SNB telling ROBIN that the login was accepted and goes through the NetBIOS rigamarole to get it delivered.

When ROBIN receives the SNB from BATMAN:

+ NetBIOS sets up a session.
+ NetBIOS gives the session an LSN. This LSN does not have to be the same as the LSN used on BATMAN.
+ REDIR adds BATMAN to the list of available servers.

At this point, an active session now exists between BATMAN and ROBIN.

SHARING A RESOURCE

Because he's a server, BATMAN knows that ROBIN will eventually want something. Servers are like parents in this regard. With that in mind, SERVER sets up a pending RECEIVE with NetBIOS.

BATMAN also sets up another pending LISTEN in case anyone else tries to log in.

True to form, the user at ROBIN now wants access to a driver resource at BATMAN. He or she types in a NET USE command to redirect a logical drive, say drive D, to BATMAN's C-DRIVE resource.

```
NET USE D: \\BATMAN\C-DRIVE
```

The NET program tells REDIR to set up the redirection.

REDIR builds ad SNB with the appropriate instructions and hands it off to NetBIOS for delivery.

When SERVER at BATMAN receives the SNB, it does the following:

+ Checks ROBIN's access rights.
+ Puts ROBIN on the list of users for the C-DRIVE resource.
+ Builds an SNB to tell ROBIN that it has access to the resource.
+ Gets NetBIOS to deliver the SNB.

REDIR at ROBIN gets this message and tells NET that the command completed successfully.

It would be nice to get a little message from NET like, "You're In" or "Roger, Wilcox" but life just isn't like that. You're returned to the command prompt. In this case, no news is good news.

FILE SHARING

Now things get *really* interesting. At workstation ROBIN, the user issues the appropriate WordPerfect commands (stand on one foot, press seventeen function keys, slap the monitor, etc) to open a document.

We'll say, for the sake of example, that this file is named "DEMO.DOC" and that it resides on drive D, the network drive.

WordPerfect makes an INT 3D call to DOS saying, in effect, "Open file D:\DEMO.DOC for reading and writing and arrange it so that other workstations can see the file but no one else can write to it. And when you're done, give me a file handle that I can use to avoid making this silly call again."

The actual line of C code might look something like this:

```
handle = open("D:\DEMO.DOC", O_RDWR | O_DENYWRITE);
```

When the network loaded, REDIR replaced the default handler for INT 3D in the interrupt vector table. This is called *rehooking* an interrupt. You've seen mention of this term if you've ever tried to remove a network driver before removing the driver loaded after it. You'll get the error, "Unable to remove REDIR. Interrupts rehooked."

Because it rehooked the interrupt, REDIR gets to see the INT 3D request before DOS. REDIR examines the request to see if the drive is on its list of network drives.

If the drive is local, REDIR passes the request over to DOS and goes back to sipping tea.

If the drive is a network drive, as in this case, REDIR starts scurrying around making preparations for the NetBIOS call.

+ First, REDIR finds the name of the server and the resource corresponding to the network drive letter. These are stored using Universal Naming Convention (UNC); in this case \\BATMAN\C-DRIVE
+ Next, REDIR builds an SNB to tell SERVER at BATMAN to open the DEMO.DOC file with the rights and access privileges specified by the calling application, in this case WordPerfect.
+ Next, REDIR builds an NCB with a SEND command to deliver the SNB to BATMAN.
+ Finally, REDIR places an INT 5C call to NetBIOS and NetBIOS sends the SNB.

SERVER HANDLES FILE REQUEST AND FILE LOCKING

When BATMAN receives the SNB, it proceeds to open the requested file and process the data for transfer. To understand this process, we need to talk about buffers again.

SERVER allocates a buffer for each workstation it services. This is the *Request Buffer*.

Net Manager sets the size of the Request Buffer via the *Request Size* option in *Server Startup Parameters*.

The default size for Request Buffer is 32 bytes. This is big enough to handle most SNBs, but anything larger, like a file write or a long DIR, will swamp the buffer and force Server to look elsewhere for its storage.

Increasing the size of the Request Buffer will improve performance for small file lookups and data transfers such as database records or DIR lookups.

In our example, the file open request is smaller than 32 bytes, so it plunks right into the Request Buffer like a cherrry in a bonbon.

Server sees the request and opens the file as soon as it can get access to DOS via INT21.

+ If internal share is disabled, Server also asks DOS to tack this file onto DOS's SHARE table.
+ If internal share is enabled, Server takes care of the locks.

In either case, the file lock prevents another user from accidentally scrambling the contents of the file while ROBIN is using it.

There's a chance that some other application already has a lock on this file. Many file and record locks don't last for long, especially in database transactions, so it would make sense for the lock request to hang around for a while waiting for the lock to clear.

It can't wait forever, though, because the application will probably start getting anxious; that is, it probably has a timeout in the tight loop it set up waiting for DOS to handle the lock and file open request.

Net Manager sets this time delay via the Lock Hold Time value in Server Startup Parameters. The default is 9 ticks. A tick is 1/18 of a second, making the time delay ½ second.

CAUTION

Some applications don't like any delay at all in getting a file or record lock. None. Nada. Nichts. Not a heartbeat.

Performance will suffer dramatically for these applications if *Lock Hold Time* is enabled.

So, when troubleshooting slow performance, try disabling Lock Hold Time and see if it helps. If not, put it back to 9 ticks or try doubling the delay to 18 ticks.

Keep in mind that lock hold time will only improve performance on a busy network because of the reduction in network traffic.

Once DOS comes back with a lock and a file handle, Server builds an SNB to give ROBIN the good news then builds an NCB to send the message across the wire and places an INT 5C call to NetBIOS.

We've been through this drill several times so from now on we'll skip the low level stuff and assume that AILANBIO is doing its job and the packets arrive safe and sound.

DATA TRANSFER FROM SERVER TO REDIR

At ROBIN, Redir gets the SNB and immediately sends back a request for the data in the file. Let's say the file contains 20K of data.

Back at the Batcave at BATMAN, here's what happens:

+ SERVER begins siphoning up data from the hard drive.
+ SERVER discovers that there is more data in the file than it can transfer directly to AILANBIO.
+ SERVER then reaches in the cupboard and comes down with a more suitable container, a Network Buffer.

That name, Network Buffer, is somewhat of a misnomer. You'd think it should be called Server buffer, but Server sets aside one of these buffers for each network task designated in Server Startup Parameters, hence the name.

Net Manager sizes the Network Buffer using the setting of the same name in Server Startup Parameters.

Additional explanation of Network Buffers

The default size of the Network Buffer is 4K. This is a small fraction of the NetBIOS messaging capacity of 64K.

A small Network Buffer forces Server to build many, many more NCBs that it would otherwise need for transfering large blocks of data.

Increasing the size of the Network buffer can improve performance considerably.

Allocating more than one Server task will also improve performance on a busy server.

There ain't no free lunch, though. Several large Network buffers take up a great deal of conventional memory.

> Example: Setting up 3 Network Buffers of 54K apiece will take 162K of memory.

Conventional memory is a valuable commodity. TSRs and applications squabble over it like frenzied holiday shoppers on the day after Thanksgiving. A user won't be overly thrilled about sacrificing a hundred and a half kbytes for something as esoteric as server buffering.

Dedicating a machine as a network server alleviates this problem. Server would be most pleased to have the lion's share of that 640K to roam around in.

A half-dozen Network Tasks, each with 54K of buffering, would not be out of the question for a heavily used server.

Once Server fills the Network buffer with as much of the file as will fit, it calls on NetBIOS to send the data across the network to ROBIN.

DATA GETS TO REDIRECTOR

At ROBIN things get a little complicated. As you might expect, the culprit is buffering.

Redirector has a buffer. Like any buffer, its purpose is to improve performance by making sure the next step in the stream has data available for processing.

When an application like WordPerfect prepares to send or receive data, it also sets up a buffer. When Redirector captures the file request, it wants to fool the application into thinking that the data is coming from a drive. Drives have buffers to store data coming off the drive, so Redirector mimics this by having a buffer to store data coming in from the network.

Redirector will only buffer data, though, if three criteria are met:

+ One: The application must open the file in Exclusive Mode.

  If the application uses another mode, a second user could modify the file while a piece of it is in the Redirector buffer. Then when the user at the first workstation saves his or her changes, they'll overlay the second user's changes and possibly corrupt the file.

+ Two: The file must be open in Sequential Access mode.

  The application must accept the data in one continuous stream from the first byte of the file to the End-Of-File marker.

Additional Explanation -- Random Access Mode

The alternative is opening the file in *Random Access* mode.

In this mode, the application can skip around from place to place in the file and pick out only the pieces that it needs.

For example, a database might have 500 records with 30 bytes per record. In the course of building a report, the database application might say, "Give me record number 200."

DOS would open the file in Random Access mode and jump to byte 6000 (the beginning of the 200th record.)

The next time the application calls the file, it might jump to record 2, then 499.

If only part of the file is in the Redirector buffer, the application wouldn't find the record it needs, forcing it to go back to the server for another piece of the file.

This initial request to the Redirector buffer wastes time, so it makes sense to bypass the Redirector buffer from the outset when accessing Random Access files.

+ Three: The data must be equal to or smaller than the Redirector buffer.

  It wouldn't make any sense to chop the data into buffer-sized portions because the excess would be discarded, which would force NetBIOS to make a second request for the lost data.

If any of these three conditions aren't met, Redirector will bypass its own buffers and put the data directly in the application's buffer.

In the case of our example, Redirector would give the entire block of data from the DEMO.TXT file directly to WordPerfect.

The following switches on the REDIR command line set the size and number of Redirector buffers:

+ BUFFERS= sets the number of buffers
+ SIZE= sets the size of those buffers

The default is one buffer of 1K size.

As with Server, more and larger buffers can help performance, but once again, the buffers take conventional memory.

AN ADDITIONAL COMPLICATION WITH REDIRECTOR BUFFERS

Redirector will not act on the contents of its buffer until the buffer is full or the originating application closes the file.

This means that a 500 byte file might sit in a 1k Redirector buffer for a long, long time, until the application closes or another batch of data comes in that fills the buffer.

A common example of this bad behavior is printing.

When an application loads, it generally opens several auxiliary files for standard devices. These include:

Console (aka, keyboard)
Cideo
COM port
Parallel port

When an application prints, it sends the output to a file named LPT1 or PRN (or LPT2 or whatever other port it's using.)

DOS recognizes any file that has the tag LPTx as a call to the designated parallel port and feeds the file to that port.

Simple. Elegant. Everyone's happy. Now enters the network.

When REDIR loads, it nabs the interrupt that controls the output to parallel ports.

When an application sends a print job to a network server, REDIR does its thing (builds SNBs and NCBs and all that jive) to send the job across the network.

Print files, by definition, are opened in sequential mode with exclusive access so if the job is smaller than the Redirector buffer (or if the application's own buffer is smaller than the Redirector buffer) Redirector will buffer the print job.

Dandy. A win all around. The app is happy because it gets shed of the print job faster so it can return control to the user. The network's happy because it can handle the data efficiently.

BAD BEHAVIOR CAUSES FILES TO HANG IN BUFFER

But what happens if the application never gets around to closing the print file? This is a common occurrence with DOS-based database and spreadsheet programs.

The answer's ugly. That last bufferful of data will sit in the Redirector buffer like the last swallow of coffee in the coffeepot. No one wants to drink it because they'll have to make more.

Remember, too, that Redirector's sending the job to Server on the networked machine. Server will keep the spool queue open until the file closes. Redirector can't close the file until the application does. Stalemate?

No. it's LPT TIMEOUT to the rescue.

LPT TIMEOUT at the workstation senses when the print job is idle and forces the file closed when it times out. This permits Redirector to flush its buffer so that Server sees that the file has closed and can complete the printing transaction.

With all these caveats and troubles, why have Redirector buffers at all? Take the case we're talking about in our example, printing from a word processing program.

+ ROBIN is running WordPerfect across the network from BATMAN.
+ WordPerfect has a rack of overlay files it uses to keep the user happy about performance. REDIR has to shuffle these overlays. Executables and overlays generally meet all the Redirector buffer requirements, so they get buffered. But any incoming data requires REDIR to flush the buffer, limiting the performance improvement. An extra buffer would come in handy.
+ Then what happens when you print? WordPerfect builds a temporary file to hold the formatted print job. This would be stored on the application server, requiring another Redirector buffer.
+ And the print job itself would be routed by REDIR back over to the print server, requiring yet another buffer to prevent REDIR from flushing and filling, flushing and filling, like a soda jerk with only one milkshake machine.

Once WordPerfect gets the file from Redirector, it goes on about its processing. The network goes back to idling, sending out the occasional heartbeat packet and keeping the sessions alive, until the application calls it again.

REVERSING THE PROCESS-- SAVING A FILE

When the user saves the file, the sequence of events essentially gets reversed with an important exception.

Before Redirector on ROBIN can send the data to BATMAN, it must first send an SNB telling Server to get ready to write to the file. Only then can it send the contents of the file in memory.

Here's where the CHAIN SEND command comes in handy. When Redirector builds the NCB for this transaction, it can point NetBIOS at both the SNB and the data. They will then go across the network as a unit. This is much more efficient than the series of Chip and Dale please and thank-yous that NetBIOS usually requires.

The NCB for this transaction might look something like the chart on the next page.

Field Name	Contents
NCB_COMMAND	17 (Command number for CHAIN SEND - NO WAIT)
NCB_RETCODE	FF (When the command completes, this FF will be replaced with the return code, which is 00 if successful.)
NCB_LSN	2
NCB_NUM	3
NCB_BUFFER@	03B100FA (The first 2 bytes, 03B1, is the segment. The next 2 bytes, 00FA, is the offset.)
NCB_LENGTH	07 (This corresponds to 112 bytes, the approximate size of the SNB.)
NCB_CALLNAME	4E2010A3038C (The first 2 bytes, 4E20, is the size of the buffer containing the data file, corresponding to 20K. The next four bytes comprise the buffer address.)
NCB_NAME	Not Applicable (BATMAN and ROBIN already have a session specified by the Local Session Number)
NCB_RTO	Not Applicable (Receive timeouts are set by the LISTEN command.)
NCB_STO	Not Applicable (Send timeouts are set by the CALL command.)
NCB_POST@	0499FAC3 (This is the address of the POST handler, which tells NetBIOS what to do when the CHAIN SEND command ends, either successfully or with an error.)
NCB_LANA_NUM	0 (The logical number of the adapter.)
NCB_CMD_CPLT	FF (When the command completes, this FF will be replaced with the return code.)
NCB_RESERVE	14 bytes reserved for NetBIOS

When AILANBIO on BATMAN gets data using a CHAIN SEND command, it treats the two buffers, one with SNB and one with data, as a single entity.

It routes both to SERVER, which plops them in a Network Buffer then sets to work writing the data to the DEMO.TXT file in accordance with the SNB.

When it receives the last packet, it sends back an SNB telling ROBIN that the transaction completed successfully.

This is the end of this module. Please send your questions or comments to the training coordinator via email.

Fundamentals of NetBIOS and LANtastic Networks

Training Workbook
Module NOS14
Revision 00
2/07/95

ARTISOFT CORPORATE MISSION

To be a leading developer and marketer of network solutions to workgroups, small businesses, and remote network users.

MISSION

To establish channel franchises through sales execution, support, and technical services on a global basis.

CUSTOMER SUPPORT DEPARTMENT MISSION

To provide such excellent service to our customers that they will recommend and repurchase our products.

EDUCATION SERVICES DEPARTMENT MISSION

To provide Artisoft Technical Support and Sales associates with the best training and best information tools in the industry.

Special thanks to Robert Mumee, Development Engineer, who generously gave his time and expertise to explain the fine points.

Additional thanks to Jim Rush, Technical Support Engineer, Lorna Peck, Training Specialist, Brad Risk, Senior Training Specialist, and Webster Hansen, Technical Support Representative, for their valuable comments and thorough review.

PLEASE NOTE: This is a training document. It is not intended for use as a troubleshooting reference. Product features and concerns can and often do change without notice. Please refer to source documents or Folios for the most current product information.