Windows NT Performance Monitoring, Benchmarking, and Tuning

- 3 -

Simulating System Bottlenecks

This chapter covers the following topics:

- **The Importance of Simulations.** This section stresses how simulating problem scenarios in your NT system can help with preventative troubleshooting and help you recognize when a real problem exists.

- **Hardware and Software Interaction.** To troubleshoot a problem in NT, you need to know how hardware and software interact. This section covers the components of a typical Intel-based system and the mechanics of how they work with one another.

- **Simulating Memory Shortages.** This section demonstrates how to simulate a memory bottleneck condition and how to isolate the specific memory problem.

- **Simulating CPU Activity.** This section discusses how to simulate CPU bottlenecks and how to isolate specific processor problems.

- **Simulating Disk Usage Conditions.** In an environment in which disk performance is crucial, it is important to test your hard drive system to determine how fast the drive media can transfer data.

- **Simulating Network Usage Conditions.** Windows NT in a network environment is subject to numerous sources of network traffic. This section details the types of network traffic and how to isolate specific network problems.

Whenever you have a problem to solve, you first try to understand the situation and then you try to change it. A step in the process of understanding and changing is experimentation. Of course, when you experiment, you use some type of scientific method. This means you record events, create controls, and run repeated and varying scenarios in an attempt to learn all that you can about a process, interaction, machine, being, or other curiosity. Computers are complex, and NT is no picnic to understand. On top of that, you have applications to worry about. This chapter is devoted to the subject of experimentation and the techniques of simulating loads and scenarios on Windows NT systems. Along the way, you will also receive some additional troubleshooting tips and bottleneck detection methods.

**The Importance of Simulations**

Simulations generally involve using techniques to create a certain situation in which you control all
but one of the pieces. Creating this kind of situation and varying some parameters allow you to draw conclusions about the way NT, a service, or an application works. In addition, creating predictable simulations gives you the opportunity to zero in on the troublemaker in the system.

When you optimize an NT system, you need a way to simulate resource usage, shortage, and various other conditions. You can achieve these simulations in various ways, depending on exactly what condition you need to simulate and what results you expect. Typically, a real-world scenario could look something like the following case study.

**Case Study**

As a systems administrator, you must evaluate a new NT database program to be used by the corporate office. Part of your evaluation is to determine whether your current system configuration is enough to run this new program. Beyond the usual "Do I have enough disk space to install the program?" question, you also face the question of how your system will respond with the new application running. Maybe you need more memory. Maybe you need more network bandwidth because it is a network program. The last thing you want to do is install the program and watch your network slow down to a screeching halt. You need to run a methodical series of tests to see just how your system reacts to the new program. You will test resources individually, from memory to network, to determine the performance impact. In addition to running the test on the individual application, you also need to test the system while it is performing its normal, everyday duties.

Why more than one test? The system will usually behave differently under different circumstances. NT itself is constantly reconfiguring itself to optimize performance based on current conditions. Perhaps the application in this example works beautifully as long as the database queries don’t generate results over 20KB. However, larger queries cause excessive delays in the data processing. This discrepancy might be due to the way the client side is producing the queries, the way the server database is constructed, or one of several database software parameters. Repeated trials will allow you to experiment and validate or reject each of these possibilities, or hypotheses.

This chapter examines how to simulate resource loads on the system and determine a few things: the impact the new load has on the system resources and how the resources are interrelated.

For instance, you will see how simulating a memory load could also affect the disk performance.

For many of these tests, you can use standard applications, as well as some utilities available in the Windows NT Resource Kit. To record the system response, you will use the Performance Monitor.

You might ask, "Why bother simulating resource bottlenecks?" The answer is quite simple: You cannot afford not to. You want to know how your system will react under pressure; "When I add those next 100 users, is the server going to crack, or does it still have more power left to handle the situation?" You should not wait until your NT system is in use before realizing that it will not handle the task it was implemented to perform. As a system administrator, it is your responsibility to make sure the system meets (and exceeds) the demands of your users. As this chapter discusses each resource bottleneck, you will learn about programs and techniques you can use to simulate resource...
bottlenecks.

Take a look at another real-life case study.

Case Study

While I was under contract with a large government organization, one of my job assignments was to determine how much load capacity the network server could handle after all 5,000 clients were using the network. The organization had built a small test environment using an NT Server and 10 NT Workstation clients. The question to answer was, "Is the NT server optimized to handle all 5,000 clients?" That is, "what is the impact on the network when all the users are logged in at the same time? What is the impact on the hard disk when users access shared files on the server? Does the server have enough disk capacity? When will we need more memory?"

These are all good and valid questions to answer. The organization did not want to go "live" without some assurances that the system would meet expectations. They wanted a realistic test drive. Without making 5,000 client connections available, how do you simulate such an environment?

The goal of this chapter is to explain how you can simulate memory bottlenecks for each resource and understand how the system responds to such problems. More important, you will also learn to use the Performance Monitor, as well as other tools, to not only track the system performance, but to also provide a different picture of what is happening in the system.

Hardware and Software Interaction

Before launching into simulations of the individual resources, let’s examine briefly how a computer is divided into a series of components that work together to provide services to the user’s applications. Figure 3.1 shows the layout of a typical Intel-based computer system.

FIGURE 3.1 Layout of the typical Intel hardware computer system.

At the heart of the system is a central processing unit (CPU). This processor has its own Level-1 cache memory and, in addition, could have a secondary Level-2 cache. In Windows NT, the system can have more than one processor, each with its own Level-1 or Level-2 cache memory. All of the processors, however, share the same physical RAM. The RAM and the CPU communicate via a high-speed 32-bit bus.

Looking Ahead

Later in Chapter 6, "Optimizing CPU Performance," you will take a closer look at the CPU architecture. You will see how the architecture of the CPU depends on not only the manufacturer, such as Intel or AMD, but also the model. For example, the memory bus and cache structure of a Pentium Pro is different from that of a Pentium II. Also, Chapter 6 has a discussion on the newer technologies that are emerging. Technologies such as SDRAM and 100MHz memory bus systems will further enhance processor performance.
Connected to this 32-bit bus is an I/O controller that is responsible for the interface of the CPU to the attached peripherals. Possible peripherals connected in this manner include hard disks, CD-ROMs, network cards, mice, keyboards, and so on. Many of these devices are connected directly to the motherboard, but many require additional interface cards, such as a SCSI scanner device, which requires a SCSI card.

In the Intel platform architecture, the I/O bus architecture can be 8-bit, 16-bit, or 32-bit. This architecture depends on the type of I/O bus that was built in to the PC. Bus specifications include ISA, EISA, and PCI. The ISA is the oldest of the technologies and typically supports the 8-bit and 16-bit ranges. The EISA is a 32-bit functionality that isn’t frequently used anymore. PCI is common and is usually built in to a system to run at 32-bit; however, the specification and technology can run at 64-bit. Examples of the interface cards that support these bus topologies are ISA, EISA, and PCI.

The CPU, memory, disks, peripherals, and I/O buses all work together to allow applications to load and run. If you look at a typical situation in which the user double-clicks an application, this is what happens:

1. The application resides on the hard disk. The mouse click informs the operating system that a click has occurred, and the OS interprets the client as a command to launch the application.

2. The application is executed. This means that the Process Manager (see Chapter 1, "Understanding Windows NT Architecture") allocates the resources necessary to run the application.

3. The Virtual Memory Manager (VMM) provides the application with up to 4GB of virtual memory to store code and data in memory. Part of the program instructions will reside in physical RAM, while the rest stay on the hard disk.

4. Parts of the program, which reside on the physical disk, are moved into memory via the I/O bus.

5. The instructions are moved from RAM to the CPU via the 32-bit bus. Instructions are stored in either the Level-1 or Level-2 cache (if present).

6. As the program executes, the VMM will return to the hard disk to load those parts not available in memory. The request will travel via the I/O controller to the hard disk device and back up to the memory.

7. The program instructions, executed by the processor, will contain instructions to do a variety of things such as draw text on the screen, write data to the network, read data from a floppy, and so on.

8. These requests will again travel through the I/O controller to the appropriate peripheral device.

The resource bottleneck can occur at any or all of these stages. The major areas that this book covers are memory, processor, disk, and network resource bottlenecks.
Keep in mind that in these eight stages, the I/O bus architecture plays an important role in system performance, especially when there is a lot of peripheral device activity. It does not matter how fast the processor is or how much memory you have. If the I/O bus speed is only 8-bit, you are effectively trying to push a lot of fast data through a slow, narrow pipe.

Recall that a component that is causing a bottleneck on a system is best defined as the component that cannot meet the demands that the rest of the system places on it. It may or may not be the slowest component on the system.

**Simulating Memory Shortages**

A memory shortage is probably the most common cause of performance problems. As you recall from Chapter 2, "Using the Performance Monitor," as memory resources decrease, the system begins to page more to the hard disk and the running applications have less memory to share. Recall that paging out is the process of moving unused information from physical RAM to a file on the hard drive called, appropriately enough, the pagefile. Also, paging in, or simply paging as I call it, is the process of bringing information not found in the appropriate spot in physical RAM from the pagefile or another location in RAM. Clearly, paging is an indication of a memory shortage or strain. As more applications run, the availability of memory resources is reduced. In turn, each running application gets a smaller allocation of physical memory. NT will fight to reduce the running application’s allocation of memory, trimming the fat so to speak as memory runs into leaner times.

It is easy to simulate a memory bottleneck. Most of the time, you can use everyday applications such as Word or PowerPoint. Just start them all at once and watch the fireworks. Most of these applications are memory intensive, especially when they first launch, so they can work for you in a pinch if you do not have other tools available. However, they will not provide "stable" memory usage. During simulations, the name of the game is control. In any scientific methodology for testing and analysis, you must control as many of the variables as possible. Simply running an application certainly makes an effect on memory. However, for a true simulation, you want a application that can use memory in predictable blocks. Also, you want to be able to adjust the application’s consumption of memory.

Some applications are better at managing memory than others. Many programs, however, have memory leaks, meaning they allocate memory but are not diligent at freeing memory when it is no longer used. The memory leak problem was more apparent in Windows 3.1 and Windows for Workgroups 3.11. These environments were not good at deallocating unused memory. For instance, under Windows 3.1, for an application such as Word you might notice 80 percent free resources available before starting and only 78 percent free resources available upon closing. After a while, that count gets lower and lower until you are forced to reboot the system. Windows NT, fortunately, does a pretty decent job of automatically freeing unused memory.

As an application uses more memory, other programs or the OS have less memory to use, thus creating a severe memory shortage. Other programs, although not leaking memory, are basically fat. They acquire and continue to use large amounts of memory while they are running but appropriately release memory when they terminate. Sometimes these memory-hungry programs can use more memory as more users start the application. This type of memory hoarding is something you want to watch.
Windows NT is efficient with memory because when an application allocates memory, the OS returns a *handle* to the memory, and NT keeps a count of how many handles have been allocated as well as the owner of each handle. If an application closes but does not free the memory it allocated, the handle to the memory becomes invalid, which means that the handle does not have a process associated with it. NT recognizes this and frees the memory in most cases. That is not to say that NT does not have programs that cause memory leaks, but they are less common than in previous versions of Microsoft operating systems.

Memory management has several other related components in the protected memory model used by NT. Applications are not permitted to interfere with other applications’ memory address spaces. One of the C2 Certification rules is that all memory should be zeroed out (cleared) prior to being allocated to another application. C2 Certification is a security rating used by the U.S. government when evaluating the situations and types of data that can be kept on computer systems.

**Memory Performance Monitoring Tools**

Now you have an idea of the types of memory problems that might occur. You must become familiar with a few monitoring techniques to properly get a practical view of memory simulations. Several tools are reviewed here to give you a first look. Later, in Chapters 6-9, which discuss the optimization of the specific NT and computer components, you will get a more thorough description of the application of monitoring techniques.

**Tracking Memory Allocation and Page Faults with PFMON**

When looking at an application (or applications) that can cause memory bottlenecks or performance problems, you should be interested not only in how much memory the application utilizes, but also in how many page faults the application causes. A page fault is generated when an application asks NT for a piece of code or data out of physical RAM, but the information has been paged out to the pagefile. The system reacts to a page fault by retrieving the information from the pagefile and placing it in the location in physical RAM where the application expects to see it. There are two types of page faults:

- **Hard page faults.** The resolution is done to the hard disk pagefile. The information sought could not be found in memory or the file system cache and thus had to be pulled from the pagefile. Disk access is clearly slower than memory access, so this process has a higher performance impact. Hard page faults are your chief concern when reviewing memory shortages.

- **Soft page faults.** The resolution of the page is still in memory but not in the location that the application expected. NT is constantly moving data around to better manage the space. Prior to moving data out to the physical hard drive, the data will pass through the file system cache. The file system cache is a section of memory used for temporarily storing information before it goes to disk. If NT can find the information here, it will move it to the proper location in physical RAM where the application can use it. Also, when the data is being shared with other applications, NT may move the information around to better optimize memory. In either case, moving data around in physical RAM has a much lower performance impact than pulling information from the hard drive. Most of the discussions in this chapter do not consider soft page faults, although enormously excessive soft page faults can still impact performance. Soft
page faults are also an indication of poor programming.

One way to look at how a particular application might cause page faults is to use the Page Fault Monitor (PFMON) Resource Kit utility, which is a command-line utility that accepts the arguments illustrated in Figure 3.2.

**FIGURE 3.2 Page Fault Monitor arguments.**

The following list provides some details on the PFMON command-line arguments:

- **-n** When PFMON is running, it displays page fault statistics and other information. This tag will suppress that information and route it to a file called pfmon.log that is created in the directory from which you launched the PFMON utility. Because this tool is a command-line argument that avoids interactive displays, it is a candidate for running in the background. You can use the NT scheduler to launch the utility when you have memory problems with particular applications.

- **-l** Normally, PFMON displays the information. If you want a record of what occurred, you use this option.

- **-c** This shows only the code faults. Code faults are page faults that occur when an application makes a call to another DLL or section of code that is not loaded in physical RAM. These types of faults can give you an indication of whether the code is written poorly. Grouping disjointed functions in DLLs or segments of code can cause such behavior.

- **-h** Remember that hard page faults are the primary mark of a memory shortage. To filter the software page faults and only look at these troublesome hard page faults, use this -h argument.

- **-p [pid]** If you want to observe a running program, you can connect to it using the -p parameter. If you simply use PFMON [application], you can have PFMON launch the application for you. In this way, you get to see all the activity that the application produces. Sometimes, however, you will not want to see this activity. When an application starts, the flurry of activity that occurs is not typical of the rest of the application’s behavior.

- **-d** This argument is exceptionally useful if you want to move the information into a database or spreadsheet for statistical analysis or simple charting. It will produce a tab-delimited log instead of the standard column format of the -l option.

- **-k** This argument will add kernel mode page faults and user mode page faults into the mix of statistics. You can see with more detail what sections are doing the most paging. If the user mode is doing more paging, the application’s code or data is typically the issue. If the Kernel mode is producing more paging, you might have a problem with a DLL from a third party or an API call.

- **-K** This argument will focus the analysis of the PFMON utility on the Kernel mode operations that the application is executing.

One of the arguments is -p [pid]. This argument enables the utility to attach to a running process. The PID is the process ID that every running process is given. You can get a list of the PIDs for each
running application by looking on the Processes tab of the Task Manager (see Figure 3.3) or by using another Resource Kit utility called TLIST (see Figure 3.4). After you have the PID for the process, you can use PFMON to monitor the page faults as well as other useful information. PFMON will list the function calls made by the program while running, display a list of the DLLs used by the program, and report a synopsis of the hard and soft page faults per DLL.

The process or application you want to monitor must be running (and have a PID) for PFMON to work. All 32-bit applications have PIDs; however, you have to watch out for those 16-bit applications. The 16-bit applications run in NTVDM.exe. The 16-bit applications do not each get a PID, but NTVDM.exe does. Thus, if you want to watch a 16-bit application, you must make sure that the application is running in its own memory space so that it is the only 16-bit application running in NTVDM.exe. This assurance is necessary for 16-bit Windows applications, but not actual DOS applications. Each DOS application will run in its own NTVDM.exe.

FIGURE 3.3 Examining process IDs on the Processes tab of the Task Manager.

FIGURE 3.4 Examining process IDs with the TLIST utility.

As an illustration of how you can use PFMON, look at an application such as Notepad and determine how many page faults it causes. The steps and figures that follow will guide you through the procedure of determining the page faults caused by Notepad.

1. Open Notepad. Notepad is available from the Accessories folder or by typing notepad from the Run command prompt (see Figure 3.5).

FIGURE 3.5 Opening Notepad from the Run command prompt.

2. Open a command prompt window. Type tlist and view the results. Record the PID value for notepad.exe. In this case, the PID value is 58 (see Figure 3.6).

FIGURE 3.6 Obtaining the PID value for notepad.exe.

3. At the command prompt, type PFMON -P and use the PID number obtained in step 2. This enables PFMON to monitor the PID number for the Notepad application. You will notice some information being displayed in the command-prompt windows. This information is a real-time listing of the various function calls made to the DLLs that are loaded. The page fault types are listed next to the function calls.

4. Return to Notepad and type something (see Figure 3.7).

FIGURE 3.7 Generating some activity in Notepad results in various lines of code and functions being called.

5. Close Notepad, saving the information to a file on the drive. Note that PFMON will also terminate, giving you summary information, as illustrated in Figure 3.8.

FIGURE 3.8 PFMON displays summary information after the application is closed. The process behind the PID ends.
Note the number of hard page faults compared to the number of soft page faults illustrated in Figure 3.8. Recall that the hard page faults are the troublesome ones. The soft page faults, although they still affect performance, are not anywhere near as detrimental as the hard page faults.

**Tracking Memory Allocation and Page Faults with PMON and Task Manager**

Another handy utility in the Windows NT Resource Kit is PMON. This command-line utility provides a good view of the memory allocation and page faults for all processes running (see Figure 3.9).

![Figure 3.9 Tracking memory allocation and page faults with the PMON utility.](image)

Notice in Figure 3.9 that the information provided is a total of the number of page faults (hard + soft) for each process, the thread count, and the amount of memory usage for both the pool-paged and the non-pool-paged memory. This tool offers the advantage of summary information for all processes on the system instead of providing the individual focus of the PFMON utility. However, it does not offer the details that PFMON does.

The same information provided by PMON is also available from the Task Manager. You can start the Task Manager in a couple of different ways--either by right-clicking the taskbar and selecting Task Manager or by pressing Ctrl+Alt+Delete and clicking the Task Manager button. Within Task Manager, select the View menu and the Select Columns option in order to select to view additional columns. Check the items you want to view (see Figure 3.10).

![Figure 3.10 Tracking memory allocation and page faults with the Task Manager.](image)

These tools are enough to get you started. Of course, the Performance Monitor is the best tool for the monitoring memory performance. That particular tool is covered in more detail throughout the book.

I suggest that you return to this chapter to refresh your memory about how other techniques and tools you have learned might be applied to of simulating activity after reading this and the chapters on the optimization of the system components.

Now, you should be ready to begin to formulate some memory simulations.

**Memory Simulation Tools and Techniques**

Now that you have a couple of tools to use to watch what is going on with page faults and general memory activity, you can learn to use some other tools to actually generate the problems. In the sections that follow, you will learn about creating memory shortages, reducing programs to minimum memory allocations, and simulating a memory leak.

Each of the tools and techniques here have a specific purpose. However, be creative! Think through what a particular application or server component is doing with system resources, and combine tools and techniques to best approximate the behavior and control your experiment.

**Reducing RAM to Simulate Memory Shortage**
The first order of business is to create a somewhat predictable memory shortage. Several reasons you might attempt to do this are as follows:

- You might want to see how a particular server configuration is behaving under a large memory load. You could either run a huge number of applications to create the load or actually fool NT into thinking that it does not have all the physical memory that is installed. This test is particularly useful when you want to correct your pagefile configuration.

- You might want to see how a particular server application operates under a memory shortage. This will give you an idea of the attention that the developers of the application gave to memory consumption. Sometimes aggressive memory usage is unavoidable, but downright gluttony is an indication of potential problems. If the developers didn’t deal with memory usage, you might face memory leaks.

- You might want to reduce the size of the memory to reduce the size of the resulting memory dump generated from a blue screen. Although this is not really a simulation for memory shortage purposes, it is worth mentioning. When a blue screen error occurs and you have settings that dump the memory to a memory.dmp file, the contents of memory will be represented on disk. This arrangement is fine if you have only 32MB of RAM so that only 32MB is loaded into a pagefile and then transferred to a memory.dmp file. However, if you have 512MB of RAM, you end up needing a 512MB pagefile and an additional 512MB of space for the memory.dmp file. Telling NT that it has only 64MB of RAM will certainly alleviate the disk usage.

Okay, so how do you do it? To simulate a real shortage of memory, you’ll run the program using less RAM. Instead of opening the computer case and removing RAM SIMMs, you can simulate less memory by editing the BOOT.INI file. The BOOT.INI file is a hidden, read-only file on the computer root directory of the C: drive, C:\BOOT.INI. This file controls the selections you have on the screen when you boot the system. Usually, you see Windows NT, Windows NT Base Video, and DOS. You can adjust the file and the command arguments to make NT ignore physical RAM and only use what you tell it to use. Create a new line in the BOOT.INI that is a copy of the normal NT boot. Then, add the command argument /MAXMEM= [amount of RAM] on the end of the line. Assuming your system has 64MB of RAM, reduce the RAM to only 16MB, as shown in Figure 3.11.

Figure 3.11 shows a new entry in the BOOT.INI file labeled LOWMEM OPTION, with the /MAXMEM value set to 16. Upon rebooting the system and selecting this new menu option, you will note that the amount of memory shown in the boot sequence is 16MB.

If you repeat the steps 1-5 from the previous example and compare the results, you will notice that the amount of page faults is higher. The system has less memory to work with, which forces more paging.

FIGURE 3.11 Simulating a memory shortage.

Windows NT Server 4.0 will become critically unstable if you restrict the amount of RAM to a number below 16MB. Because this is the lower limit of acceptable behavior, there is little value in testing below the 16MB boundary. On Windows NT Workstation 4.0, 12MB is possible--but you won’t be happy with the performance, even for testing.
Finding the Minimum Memory Allocation

Suppose you want to find out the minimum amount of memory each application is going to take. The amount of memory that an application needs for its code and data is called the *working set*. When you push the application down to its lowest level, you get the minimum working set for the application. An NT Resource Kit utility called CLEARMEM.EXE is exceptionally adept at reducing the applications and services running on NT to their minimum working sets. You may be wondering about the value of knowing the minimum working sets for applications. Knowing the minimum working sets for applications provides you with a baseline for further analysis, subsequently enabling you to quickly identify memory-hogging applications.

Often in troubleshooting performance issues, it is difficult to tell exactly when you have a problem. You must first establish a reference point. The reference point is typically called the *baseline*. In this case, you are establishing a baseline for the applications. When you want to see how applications are responding to a memory shortage, you can compare the minimum working set size (represented in bytes) to the size of the working set in a simulation that creates a memory shortage.

If the minimum working set for a particular program is much larger than the others on the system, you might consider that program a little fat (or memory intensive). Programs can be large for several reasons, not all of them bad. Certainly, poor programming with little attention paid to memory efficiency can make a program bulky. However, in other cases, you might want more code and data loaded into memory for a specific reason.

Case Study

For example, say you are writing a piece of an application for a fuel company. The application is written for NT Server, which is interfaced with sensitive fluid pressure measurement probes. The application is built to dynamically adjust the flow of fluid through pipes to control the pressure in the pipes. The application must respond in real time to information from the probes. If the system does not respond quickly enough, the pressure in the pipes could damage the pumping equipment or even burst a pipe. In such a situation, your code must run in real-time mode or close to it. With such a single-minded task for the NT Server, you probably want to load and keep in memory the code for your application. This arrangement makes the program’s minimum working set unusually large compared to normal programs; however, it also improves the performance of the application because all the components and functions are always in physical RAM.

You might also want to identify applications with memory leaks. Recall that a memory leak is a situation in which an application uses memory but fails to properly free the memory when it is done. You can find out which application has the memory leak problem by observing the minimum working set of an application when it first starts. Then, after waiting some time, you can observe the minimum working set again. If the value increases, the application has some type of memory leak. You then must evaluate whether this leak is going to present a problem for you. A user application with a memory leak is not as bothersome as a service or driver with a memory leak. A user application will terminate, and users will usually log off the workstation at some time. When this occurs, the memory for all applications in a user’s process returns to the memory pool of the system.
If the memory leak is from a service or driver, the problem has more potential to disrupt operations. The services will not have the memory reset unless the system is rebooted. Thus, they continue to grow in size. It is like the movie *The Blob*; the services or drivers will simply continue to grow into a large glutinous oozing mass of memory, sucking code until the system crashes due to a critical lack of memory. Not a pretty picture. This process can take hours or even days because memory leaks can be very small.

Determining the minimum working set actually requires two tools. You use CLEARMEM to force the applications to their minimum working sets and good old Performance Monitor to observe the value of the applications’ working sets. Prior to running CLEARMEM, you need to make sure that you have a pagefile equal to the size of physical RAM at a minimum. CLEARMEM is going to force all the process’s code and data out of memory, and the pagefile is the destination. The following sequence shows the steps in performing this activity.

1. Start the Performance Monitor first. CLEARMEM.exe causes a lot of activity on the system when it first starts. You won’t want to start Performance Monitor while CLEARMEM.exe is forcing applications (including Performance Monitor code) out of memory.

2. In Performance Monitor, alter the type of display to a histogram view. Click Options from the drop-down menu and select Chart. Then, select Histogram and click OK (see Figure 3.12).

   **FIGURE 3.12** *Adjusting the Performance Monitor Chart view to display a histogram view of the process’ working sets.*

3. Now you need to add objects and counters to the Performance Monitor to view the activity. You can press the plus sign (+) on the toolbar to add objects. Select the process object. The list of instances will appear (see Figure 3.13). If an application or service is not running at the time you add the object to the Performance Monitor, you will not see it in the list of instances. Make sure that any and all applications, services, and drivers are installed and running at the time that you do this. In the Counter box, select Working Set. In the Instance box, select all running instances. Then, click Add. Click Done.

You will notice an instance called Idle. This particular item is not really a process but a placeholder. The system runs the Idle thread on the process when nothing else is going on. CPUs always like to be busy running something. Chapter 6 covers this concept in more detail.

   **FIGURE 3.13** *When adding the process counters and objects, you will see a list of all processes currently running on the system.*

4. You are now ready to start the CLEARMEM program. Open a command prompt, and go to the directory where you have CLEARMEM installed. Now, type `CLEARMEM.exe`. When CLEARMEM.exe launches, the system will temporarily suspend other activity; this is due to CLEARMEM running several high-priority tasks. CLEARMEM will display in the DOS screen the status of the operation. As soon as it completes, run it a second time to make sure that all applications have the minimum working set in place.
5. You can then observe your findings in the Performance Monitor window. The display will look similar to Figure 3.14. If you have the Server Services set to Maximize Throughput for Network Applications, you might need to run the CLEARMEM a third time to make sure that the services are properly reduced to the minimum working sets.

You will probably notice that the Performance Monitor has a larger working set than most of the other inactive applications. The Performance Monitor is actively attempting to collect data, so its working set is larger than programs that are primarily waiting for user input.

FIGURE 3.14 Performance Monitor displays the minimum working set sizes in a Histogram view after CLEARMEM has run twice.

Simulating the Memory Leak

You have already learned what a memory leak is. Now you will learn how to simulate a memory leak or any other slow consumption of memory. The NT Resource Kit Tool called LEAKYAPP will slowly consume memory on a system until it is almost completely used. It is like CLEARMEM in that it will reduce the applications to their minimum working sets. However, LEAKYAPP is more relentless in its memory hoarding. It will not allow other applications to retrieve the memory after LEAKYAPP has acquired it. Thus, the system must page information in and out of the pagefile. Why would you want to wreak such havoc on your system?

- You can learn a great deal about the configuration of your system and the ability to handle critical memory shortages by creating a memory shortage using LEAKYAPP. You can see how well the system you configured interacts with the pagefiles. Then, you can tune the system for better performance under these conditions.

- You can see what a memory leak looks like. What better way to diagnose a problem than by investigating it under controlled circumstances?

- Because LEAKYAPP has finer control than the CLEARMEM, you can view the system behavior at various stages of memory stress. LEAKYAPP is a graphical tool that allows you to stop and start the consumption of memory. You can pause the consumption at any time. LEAKYAPP will not release the memory until the application is terminated, so the system has to run with what it has left.

To run LEAKYAPP, you first prepare the NT Performance Monitor. You also want to make sure that you have a suitable pagefile (RAM + 12MB). Then, follow the procedure that follows:

1. Open Performance Monitor to the Chart view.

2. Click the + to add counters to the Performance Monitor view. You will add the Pages/sec and Available Bytes counters from the Memory Object.

3. Click Done. Although you use the Pages/sec and Available Bytes counters just for starters, you can add other counters, depending on what you want to observe. For example, you might want to use the steps in the CLEARMEM example, where you add the working sets to view...
how the individual programs react to the slow reduction of memory. You might want to add the
%Usage counter from the PageFile object to see how the pagefiles you set up are being utilized. You might want to look at any number of disk object counters to see how your disk subsystem is reacting to the excessive paging. The disk counters are covered in detail in Chapter 8, "Optimizing Disk Performance."

4. Start the LEAKYAPP application. You see the interface display in Figure 3.15.

**FIGURE 3.15** LEAKYAPP has a graphical interface that allows you to control the application’s consumption of memory.

5. Click the Start Leaking button. LEAKYAPP starts using up the memory, forcing NT to push information out to the pagefile. The pagefile usage is tracked on the LEAKYAPP interface.

As the LEAKYAPP continues to use up memory, NT will continue to push information out to the pagefile. You will see the Available Memory counter in the Performance Monitor slowly degenerate. NT will deal with this reduction in memory. You will see paging increase, but perhaps not as much as you might expect (see Figure 3.16). The paging does not become a big problem until the available memory drops below 4MB, an important limit for NT.

**FIGURE 3.16** The Performance Monitor shows that NT is able to deal with the reduction in memory until it reaches 4MB.

NT attempts to maintain the 4MB of available memory for shuffling information back and forth between the various locations of file system cache, physical RAM, and pagefile. You will see NT dip into this 4MB as memory becomes very scarce. This indicates that all processes are at their minimum working sets and the file system cache has been reduced to its minimum size.

Now you are ready to simulate a multitude of memory situations on NT. Remember, you can use each of these scenarios in conjunction with the others to offer different twists on themes. Be creative when applying the simulations techniques. You are now ready to move on to CPU simulations.

**Simulating CPU Activity**

A CPU is the heart of the system. The main responsibilities of the CPU are executing program instructions (which can be from the OS or applications) or servicing hardware requests (interrupts). Because a CPU can only perform one instruction at a time, the performance depends on how many instructions the CPU can do in a specific amount of time. The CPU architecture plays an important role. RISC (Reduced Instruction Set Computers) processors can process more instructions per second than the CISC (Complex Instruction Set Computers) processors common to Intel-based hardware.

With CPU architecture changing so rapidly, it is difficult to keep up. In addition, books about CPU architecture tend to delve far too deep into the realms of electronics, and CPU chip developers maintain several theories governing queue handling, cache coherency, and other concepts. At this point, you could use more information about CPU/software interrelations. A good place to look is the manufacturers’ Web sites. http://www.compaq.com has my vote as one of the best sources of information. Visit the support section at that site. After you select a server, workstation, or some other
technology, you will see a list of support documentation and technical white papers. A site search will always turn up some type of document. You can also find information on the Intel Web site, although it seems that Compaq has a little more information on the integration of technology with systems and software.

The CPU is responsible for executing all the instructions for each of the running applications and for the operating system. These applications come in many flavors and do many things, such as draw bitmaps on the screen, request services from the network, or request information from the hard disk.

Recall from the beginning of this chapter that the CPU has its own Level-1 cache, and, depending on the CPU type, it can have an external or additional Level-2 cache. The CPU is connected to the physical RAM using a 32-bit bus on a typical workstation. However, the memory bus speed will depend on the processor architecture. When the program executes, the program is loaded, for instance, from the hard disk into RAM (not the whole program, just a part of it) and then from RAM to the CPU, where the instructions are executed. The L1 and L2 caches store the most recently executed commands, and the cache memory is much faster than standard RAM (but also more costly).

The Windows NT applications running on a system will no doubt place some form of "stress" on the CPU. This stress comes in the form of instructions to execute. For a multithreaded operating system, the CPU is also responsible for managing the various threads of execution generated by the operating system and other multithreaded applications. A thread is a small and distinct set of instructions that the processor must respond to. The thread is the smallest particle of the programming element.

An application always has at least one thread of execution--the main schedulable thread. Remember, the process itself is not a schedulable entity. It is that initial thread of the program that is scheduled by the OS to run on the processor.

Many Windows NT applications will run many threads. You can see how many threads your NT system is handling by using the Task Manager and making sure the Thread Count option is selected in the Columns menu, as shown in Figure 3.17.

**FIGURE 3.17** Querying the number of threads being handled by NT.

Unfortunately, many applications are not written properly and, in fact, create an excessive number of threads. If the program creates a large number of threads at the same time, the NT Microkernel component called the scheduler or dispatcher will be forced to manage them all at the same time. Think of a ball juggler juggling three balls. A good juggler can handle three balls with relative ease. Imagine someone throwing five more balls at the juggler. All of a sudden, the juggler (in this case, the scheduler) has a lot more work to do. Instead of three balls (CPU tasks) to juggle, it has eight. This juggler must truly be the best juggler of all time. At the time the additional five balls are added, the juggler becomes busier but eventually gets it under control. In other words, the work gets busy for a moment. The same thing happens when a program "launches" many threads or processes at the same time. The scheduler will get extremely busy trying to schedule the threads.

Do not confuse the NT Microkernel thread scheduler with the Scheduler Service that runs on NT. The Microkernel scheduler controls the running of threads on the processors. The Scheduler Service is a service on NT that allows you to run programs at regular intervals, such as a backup program running every night.
Threads are more than balls in the air, however. Each thread carries with it a priority. These priorities determine which thread will get run on the processor and when. A thread of higher priority will get more chances to run on the processor than a thread of lower priority. In Chapter 6, you will be exposed to more details of this process and the theories that govern it. For now, you only need to understand the simple concept that a higher priority means more chances to run on the CPU. You will simulate the act of "throwing a lot of balls" at the CPU and see how it reacts. You can use another Resource Kit tool called CPU Stress to accomplish this. This program generates the activity needed by creating threads of a variety of priorities. You will monitor the CPU performance using the Performance Monitor. The CPU Stress program lets you create four threads for which you can select the amount of activity, as well as the priority. Using CPU Stress makes it easy to simulate thread activity, as shown in the following steps:

1. Start CPU Stress, which resides in the PerfTool\Meastool directory where the NT Resource Kit is installed. Notice that Thread 1 is already set to active (see Figure 3.18).

**FIGURE 3.18** CPU Stress is a simple graphical tool with the single-minded task of generating activity for the processor.

2. Start the Performance Monitor. Select to monitor the Processor Object: % Processor Time and Interrupts/sec and System Object: System Calls/sec. Watch the activity generated (see Figure 3.19).

3. Click to activate all four threads in CPU Stress and set the activity of all the threads to busy. Note the increased activity and how the %Processor Time counter increases almost to 100 percent (see Figure 3.20). This means that the system is now CPU bound.

How can you use this information to plan your system? Although it is hard to predetermine all the applications that are going to be running on the system, you can still use these simulation tests to understand how the CPU responds to thread activity. In this process, you are also monitoring the System object’s System Calls/sec and the Processor Object’s Interrupts/sec. The Interrupts/sec counter relates to the number of times the CPU is interrupted by hardware devices to perform some task. System Calls/sec relates the number of calls made by the OS to the CPU. In the test scenario here, the increase in activity is caused by system calls and not a hardware device.

**FIGURE 3.19** Performance Monitor is where you examine the effects of CPU Stress on the processor.

**FIGURE 3.20** CPU Stress causes the processor to become almost 100 percent utilized.

---

Use the Processor: Interrupts/sec and System: System Calls/sec counters to determine whether the CPU performance problems are software- or hardware-related. This particular tactic along with some further information is presented later in the Chapter 6. In addition, the concept of a Deferred Procedure Call (DPC) is discussed. For now, think of a DPC as an interrupt whose processing is set aside until a later time.

Using the CPU Stress tool is the best way to isolate and manage the CPU performance roughly independent of the rest of the system. Now, let us move on to disk usage.
Simulating Disk Usage Conditions

In addition to the operating system, the hard disk also contains the application programs, dynamic link libraries, and data files. Poor disk performance is characterized by poor I/O performance. Figure 3.1 outlined the standard layout of a computer system with a CPU, memory, and peripheral devices. Because the disk is strictly an I/O device, you want to be able to test how fast you can read from the device and how fast you can write to the device. Obviously, the faster the reads and writes, the faster performance is overall. Good disk performance leads to faster program loads and executions.

Good disk performance has other advantages as well. First, it improves memory performance. Because the pagefile resides on the physical disk, every time the system has to page, it will obviously go to the hard disk for resolution. If the disk is slow, the paging process becomes the slower, consequently decreasing system performance.

Another benefit of good disk performance is that printing speed is faster. The spool file created by the printer is created on the hard disk. The faster the output from the disk to the printer, the faster printer output is achieved.

Keep in mind that each of these processes use the same resource—the computer’s hard disk or disks. The disk systems, much like the processor, are servicing a queue. A queue in this case is a stack of requests to put and get information from the hard drive. Think of the inbox in your email program. People need you to do something; they send it to you. People need information from you; they send a request and you respond. Managing the queue can be quite a task, as anyone who gets more than 20 emails a day can tell you. Exercising how the entire disk subsystem means stressing the NT I/O manager, the disk controller, and the physical hard drive. Throughput and performance relate to how well these components work together to service the disk queue. For this reason, it is important to understand the performance of these components under specific conditions. No matter what the disk seek time might be, it depends on how fast the controller can move information to the disk.

If the system will be using the hard disk intensively—that is, many application programs are executed and issue many I/O disk requests to files—then it is important to understand the overhead associated with each I/O transaction. When you purchase a hard disk, you are given some performance numbers, such as seeks per second or data transfer rates. What are the numbers in a real-world scenario? A typical question you might ask is, "If I create a 40MB file, how many I/O operations will it take to fill the file with data and read it back?"

Is this information really important? Many people know to get the fastest disk available. For others, this information is crucial in the design of their domain system. This is especially true when the system will be used primarily as a file or print server and the hard disk will be accessed frequently. The system might serve as a database or FTP (File Transfer Protocol) server, with many users uploading hundreds of files.

Now that you have an understanding of the importance of simulating and testing disk performance, let’s get a handle on how to do it.

Activating Disk Performance Objects
As with other measurements, you will use the Performance Monitor for data collection. Remember to run the diskperf utility to activate disk performance objects on the computer. To do this, open a command prompt and type `diskperf -y`, and then shut down your computer and restart. Make sure to wait a while after you reboot the system before beginning your disk drive tests. When NT starts up, many background initialization activities can interface with the measurements. If the computer you are using is connected to the network, it might be a good idea to disable the network drivers. Some drivers respond to network traffic and events even if they are not directed to your computer, creating unwanted activity.

You must be a member of the Administrators local group to run diskperf.

What exactly does diskperf do? The diskperf program installs a special device driver called the disk performance statistics driver, which gets added to the stack of drivers managed by the I/O Manager. Figure 3.21 shows where the driver is placed on the stack.

![Figure 3.21](image)

**FIGURE 3.21** *Diskperf will add a driver to the stack of drivers for the disk I/O controlled by the NT I/O Manager.*

Unlike the other performance counters, the disk counters require a manual switch because they do not permanently enable performance monitoring. This decision by Microsoft was a result of the effects of monitoring the disk subsystem on an i386 machine. The 386MHz Intel processor architecture experienced a 1.5 percent performance hit when the disk counters were enabled. Microsoft felt that the user community would consider this unreasonable. It has not changed this counter, even though the performance hit on a Pentium machine is less than .05 percent. With proper disk controllers and disk caching on the controller, this value drops off into oblivion.

If you are still concerned about the performance hit on your system’s disk subsystem, regardless of how small, you can turn off the disk counters. At a command prompt, type `diskperf -n`. This will disable the counters when you reboot.

If you are using a software stripe set or software strip set with parity, you need to run diskperf -ye instead of only indicating the -y argument. This will put the appropriate counter driver in place.

Now that you have activated the disk performance counters, you are prepared to begin a simulation.

**Testing Hard Disk Performance with Response Probe and Performance Monitor**

In this case, you will use a special tool from the Resource Kit designed to simulate all sorts of activity, the extensive Response Probe simulation tool. As discussed at the beginning of this section, you can test the performance of the disk by creating a file and determining the disk performance numbers during the file reads and writes. The Response Probe will perform this activity and many others. The tool consists of an executable and some DLLs for execution. The remainder of the tool is a combination of text files that control the execution of the simulation.

**The Response Probe Simulation Tool**
Before continuing, let’s take a little closer look at this tool. The Response Probe attempts to simulate activity that programs or a user might create. It does this by setting up a group of processes and threads that track performance based on a bell-curve distribution you can control. The script files that Response Probe uses will determine how many times the processes and threads are executed and determine the bell curves that represent the distribution of activity performed by each thread. Each set of script files consists of the following:

- **.SCR** The main Response Probe file that outlines which process script file to run and how often.
- **.SCP** The process file that determines which thread file to run and how often.
- **.SCT** The thread file that details the bell curves in the form of standard deviation and means that govern the activity of the response probe simulation.

Of the three files, the .SCT file is the most important for simulation purposes. Look at the following diskmax.sct sample from the NT Resource Kit:

```
# Diskmax Thread Description File
#
# Description:  This script file is part of a test of maximum disk throughput.
#               It creates a single threaded process that does reed sequential,
#               unbuffered reads of 64K records from a 20Mb file.
#
# Format:
#               THINKTIME       Mean  SDev      (milliseconds)
#               CYCLEREADS   Mean  SDev      (number)
#               FILESEEK      Mean  SDev      (records)
#               CPU TIME      Mean  SDev      (milliseconds)
#               DATA PAGE     Mean  SDev      (page number)
#               FUNCTION      Mean  SDev      (function number 1-1000)
#               FILE ACCESS   fileaccess (file name)
#               FILE ATTRIBUTE fileaccess      (file name)
#               FILE ACCESS MODE  BUFFER | UNBUFFER | MAPPED
#               RECORD SIZE   number_of_bytes (default - 4096 bytes)
#               FILE ACTION   R | W
#
# Mean  Sdev
#
THINKTIME  0    0         No think time.
CPU TIME    0    0         No other processing time.
CYCLEREADS  100  30       Reads 100 times/cycle on average, with a s
FILESEEK    0    0       Fileseek is ignored for sequential access.
DATA PAGE   0    0       No datapage activity.
FUNCTION    500  0       Reads the middle function repeatedly to simulate
FILE ACCESS  WORKFILE.DAT Reads the 64K records from this file.
FILE ATTRIBUTE fileaccess      (file name)
FILE ACCESS MODE  UNBUFFER MAPPED
RECORD SIZE   65536     Reads directly from disk without the file system
FILE ACTION   R         Reads 64K records.
```

This particular simulation is nicely commented. Several other examples in the PerftoolProbe\Samples directory demonstrate several features of the Response probe.

One item, THINKTIME, prompts an interesting discussion into how Response Probe works. The
Response Probe is not simply copying files back and fourth; it can make an attempt to simulate various user activities. The Response Probe goes through execution in three stages:

- **Think Time (parameter THINKTIME).** This is where the system pauses. This simulates the system waiting for the user to respond to information, such as the results of a database query in an application.

- **File Access Time (parameter CYCLEREAD).** Response Probe now takes the time to access a file based on the other parameters.

- **Compute Time (parameter CPUTIME).** Response Probe will use some of the CPU’s time based on this parameter.

Together, these stages outline the type of system behavior Response will simulate. Notice that each one has a mean and a standard deviation. A review of basic statistics: The mean is the average amount of time given to each process; the standard deviation describes how far off the mean the time is allowed to wander. It is much like deciding it takes 30 minutes to get to the airport, plus or minus 10 minutes. The mean is 30 minutes and the plus or minus could be considered one standard deviation. The standard deviation basically determines the maximum and minimum values to be expected. Most values (over 90 percent) will fall within three standard deviations from the mean. The example has a standard deviation of 30 with a mean of 100. Three standard deviations are $3 \times 30$ or 90. Thus, you can expect to spend anywhere from 10 reads/cycle to 190 reads/cycle. A cycle is determined through the speed of the processor. Now, let's get a little closer to the physical world.

Because the operating system is designed to use the file cache before going to the physical disk, you must set up the test to bypass the cache and go directly to disk. For this, you need to set the FILEACCESSMODE parameter in the Response Probe to UNBUFFER, as in the diskmax.sct example.

**Applying Response Probe Disk Simulation**

The Resource Kit's Response Probe is an ideal tool for testing the response of each system resource. You can also use this tool to collect baseline performance information or to determine the maximum throughput values. The ideal use for optimizing an NT system is to use Response Probe to predict how your system will handle different workloads and how changes you have made to the system have affected the overall system performance.

In this case, you want to test the disk subsystem's performance. Follow the steps outlined in the following sequence to complete this task:

1. Create a new folder and put the Response Probe programs and data in this new folder. In this case, assume the folder is called C:\DISKMAX. Copy the following files from the folder where you installed the NT Resource Kit to the folder C:\DISKMAX:

   \PerfTool\Probe\Creatfil.exe

   \PerfTool\Probe\Probeprc.exe

   \PerfTool\Probe\Probe.exe
2. You now need to create the file. For this sample scenario, create a file that is 20MB in size. You do this with another NT Resource Kit utility called CREATEFIL. CREATEFIL creates a file of a particular size for use with Response Probe. You need to make sure that the name of the file matches the name you defined in the .SCT file with the FILEACCESS parameter. In the case of the DISKMAX sample, you will use the filename workfile.dat. Open a command prompt and set your directory to the C:\DISKMAX directory you created in Step 1. To create a file, begin by typing the following:

```
createfil workfile.dat 20000
```

The workfile.dat file created is a 20MB file filled with zeros. Response Probe will use this file during its simulation to create a workload for the disk.

3. Next, you use the available disk examples in the Resource Kit. They are located the RESKT\PERFTOOL\PROBE\EXAMPLES directory. The files of interest here are the three DISKMAX.* files:

DISKMAX.SCP (the thread definition file)

DISKMAX.SCT (the thread description file)

DISKMAX.SCR (the process file)

You copied these files to the C:\DISKMAX directory in step 1.

4. You also need to use the Performance Monitor to monitor the disk activity. For this, you’ll monitor the LogicalDisk object illustrated in Figure 3.22.

**FIGURE 3.22 Adding the LogicalDisk object to the log file for monitoring purposes.**

Instead of using the Chart view, you will log the data. Logging the data will give you the opportunity to examine it later. The specific counters you want for the LogicalDisk object follow:

- **Avg. Disk Bytes/Read.** The average number of bytes transferred from the disk during read operations.

- **Avg. Disk sec/Read.** The average time (in seconds) it takes to read data from the disk.

- **Disk Read Bytes/sec.** The rate at which bytes are transferred from the disk during read
operations.

- **Disk Reads/sec.** The rate of read operations on the disk.

Set the Performance Monitor to log the information to a file at an interval of 15 seconds in the Periodic Update box, as shown in Figure 3.23.

**FIGURE 3.23 Setting the Performance Monitor logging interval.**

5. You are now ready to start the simulation. Use the Probe.exe program to process the script file. The syntax follows:

```
Probe diskmax.scr 600 diskout.out
```

You'll use diskmax.scr as the name of the Response Probe script file. The 600 refers to the total test time in seconds, and diskout.out is the name of the output file. This output file name is optional. The default output filename is the same as the process script file with the .OUT suffix.

The Response Probe does not actually begin the test immediately. It waits about one half of the test time before measuring any values. The simulation has this delay while the system quiets down. Resource Probe is the same as other applications in this regard; the resource usage during startup is not exemplary of the system usage during regular runtime.

Figure 3.24 shows the collection taking place while the Response Probe is processing the script file.

**FIGURE 3.24 Data collection activity occurs while the Response Probe is running and the Performance Monitor is logging information.**

After the Response Probe completes its operation, you will have a Performance Monitor log that contains information from the entire simulation. Using any one of the Performance Monitor views (such as Chart or Report), you can load this log file and review the information.

Figure 3.25 shows the LogicalDisk counters using the Performance Monitor report format. To view information from a Performance Monitor log, access the Chart, Alert, or Report views and click Options from the drop-down menu. Then, select the Data From option. You will be prompted for the location of a file. Figure 3.25 shows the disk throughput denoted by the Disk Read Bytes/sec value.

**FIGURE 3.25 Using the Performance Monitor Report view, you can see the information collected during the simulation.**

This sample simulation indicates that the disk throughput is 5263381 bytes/sec. Whether this is good depends on the disk subsystem architecture, the seek time on the disk, and the controller. For an ISA EIDE drive of decent power, this number is excellent. For a SCSI device on an ISA controller, it is good. For a SCSI device on a PCI controller with plenty of RAM on the controller, you might need to wonder what is wrong. More details about how the architecture affects throughout is available in Chapter 8.

Take a look at the other parameters. The Disk Reads/sec tells you how many times the disk was read
to satisfy the requests for information. A decent I/O system is capable of about 40 I/Os per second. This report shows an impressive 80 I/Os per second, indicating a good controller. The Avg Disk secs/Read is a value that you compare to the seek time the manufacturer reported for the drive. Under the best conditions, the values should be close.

Using the Response Probe, you can simulate other types of activity. For disk activity, reporting that this one trial is a complete reflection of disk performance would be a mistake. The disk subsystem is servicing a queue, so it is subject to changes in how data is placed in the queue. The disk is also susceptible to changes in the size of the file reads. When the reads match the sector size, life is good. When they don’t, life gets a little harder for the disk. In the example, the files were read sequentially. If you modify the test to use random reads, the performance of the disk will drop. Random reads mean the disk must spend more time moving the reader arm around on the physical disk platen, which takes time.

**Simulating Network Usage Conditions**

In an NT domain, the network could be considered the "veins" of the system. Information is carried to and from the network, providing a conduit for network traffic, such as logon validations, resolution of NetBIOS names, handling of DHCP requests, file transfers, downloading of Internet pages, and directory replication. As with the blood in your veins, you take the actions of the network for granted. You know the network is functioning; you just don’t see what is being sent across it. Like your veins, the network can get clogged when all of a sudden, the data is not being pumped as fast as it used to be.

You can create a stress load on the network in several different ways. If you really want to simulate the effects of a SAM database replication between the PDC and the BDC, for example, you need to capture that event. The best tool to use is a network sniffer, such as the Microsoft Network Monitor. The Network Monitor and its usage are covered in detail in Chapter 9, "Optimizing Network Interface Performance."

If you want an easy way to determine how the network handles network traffic, however, it can be as simple as creating a large file and transferring it across the network. This methodology may be a bit crude, but it does do what you want: put a load on the network. If you need to replicate a 1MB file from one machine to the next every hour, you could create a 1MB file and copy it manually while looking at your network performance.

In this case, you do not necessarily need to use the Network Monitor. You can use the Performance Monitor, provided the Network Monitor’s extended objects are installed. You can install them by installing the NT Server version of the Network Monitor tools. From the Network icon in the Control Panel, select the Services tab. Then, click the Add button and select the Network Monitor Tools. Install, and, when prompted, reboot. You will need the NT distribution CD-ROM to complete this operation.

When you reboot and you start the Performance Monitor, you see an additional object called the Network Segment. This object contains numerous counters, including % Network Utilization, which is what you will use for testing purposes in this section.

Now you need a large file to test. You can use the CREATEFIL utility from the Resource Kit. Create a 10MB file:
To make it easy, create a recursive batch file that copies this file to another computer, deletes it from the current machine, copies it back to the computer, and deletes it from the remove computer. The batch file would look like this:

```
REM NWLOAD.BAT
Copy c:\tempdir\testfile.tst \remote\c$\testfile.tst
Del testfile.tst
Copy \remote\c$\testfile.txt c:\tempdir\testfile.tst
Del \remove\c$\testfile.txt
Call nwload.bat
```

The sole purpose of this batch file is to create network traffic by copying this large 10MB file back and forth across the network.

Now, open the Performance Monitor and look at the % Network Utilization counter of the Network Segment and the Current Bandwidth of the Network Interface object. From a command prompt, start the special batch file that you just created. You should see the Network Utilization counter increase but the Current Bandwidth counter remain constant, in this case, at around 10. Why? You are using a 10BaseT network with a maximum bandwidth of 10 megabits per second.

Assume that the % Network Utilization counter is at a constant 85%. This generally indicates that the network bandwidth is being fully utilized. Consider the following crude but illustrative calculation:

**Network speed: 10Megabits/sec**

Convert to Megabytes:

10 Megabits/sec ÷ 1 Megabyte/8 Megabits = 1.25 Megabytes/Sec

Transfer Time for the File:

10MB file ÷ 1.25MB/sec = 8 seconds

This calculation tells you that for eight seconds, the file transfer will use 100 percent of what the network has to offer. However, the value that was displayed in the Performance Monitor was only 85 percent. A network can't really reach the 100 percent mark. The network will become saturated, and the transmission of additional new packets of data will be inhibited due to collisions and retransmissions on the network.

**Summary**

You can use the techniques in this chapter to apply a resource load to the system and simulate a bottleneck. These techniques will also help you understand what happens when the tested resources are in short supply and high demand.

More importantly, you now know the various Performance Monitor counters that you can use to understand how the system is affected by the simulated workloads. Revisit the case study presented at
Case Study

As a systems administrator, you must evaluate a new NT database program to be used by the corporate office. Part of your evaluation is to determine whether your current system configuration will be enough to run this new program. Beyond the usual "Do I have enough disk space to install the program?" question, you also face the issue of how your system will respond with the new application running. Maybe you need more memory. Maybe you need more network bandwidth because it is a network program. The last thing you want to do is to install the program and watch your network slow down to a screeching halt.

In this case, you need to determine the impact of this new program on the system with relation to the memory, for instance. The steps you could take follow:

1. Determine how many page faults the new program generates. Use the Resource Kit PFMON utility as demonstrated earlier in the chapter.

2. Run this application along with the Performance Monitor to determine whether your application is CPU intensive by monitoring the % Processor Time counter.

3. You can determine if the normal operation consumes excessive network bandwidth by monitoring the % Network Utilization counter.

4. You can determine the impact on the disk resource by monitoring the LogicalDisk object.