Inside Microsoft Terminal Server

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

NT Technology

As with many technologies, what's new in computing is often old, and the trend toward empowering users with high-maintenance, powerful PCs is weakening. Many IS managers are realizing that centralized management with a single mainframe and users connected via thin clients has advantages over a sprawling client/server installation. With centralized management, systems administrators can maintain a few servers and minimal-intervention clients instead of worrying about administering and upgrading software across hundreds or thousands of computers.

Microsoft designed Windows NT in the early 1990s for single-user workstations that run demanding standalone and client applications and for file and print servers that might also run disk-intensive server-side applications (e.g., database engines). Citrix provided support for application- and CPU-serving in late 1995 through an independent version of NT 3.51 called WinFrame. In May 1997, Microsoft and Citrix signed an agreement to cross-license their technologies so that Microsoft could bring WinFrame's thin-client support into the mainline NT code base. Microsoft plans to release the commercial version of its thin-client solution (formerly code-named Hydra) as Microsoft Windows NT Server 4.0, Terminal Server Edition, by July 1998. Citrix has a product, MetaFrame (formerly pICAsso), with features that enhance Terminal Server's features. MetaFrame supplements, but doesn't replace, Terminal Server. (To read a recent review of the beta 1 versions of Windows-based Terminal Server and pICAsso, see John Enck, "Spawn of the Hydra," March 1998).

In this column I'll give you an overview of Terminal Server and describe how it differs from standard (non-Terminal Server) versions of NT. I'll describe the parts of NT that Microsoft has modified or augmented to support multiple users. Often, these modifications involve subtle changes in dozens of NT's modules. However, I'll highlight only major architectural changes and examine the form and functionality of Terminal Server. If you don't have a solid background in basic NT architecture, I recommend that you read my March 1998 and April 1998 NT Internals columns for an overview of NT's architecture.

Terminal Server Overview

Terminal Server's goal is to make NT applications available to users operating inexpensive thin-client terminals. With respect to Terminal Server, a thin client is any machine capable of displaying at least VGA-quality graphics (640 * 480 resolution with 16 colors), processing input from a mouse and keyboard, and sending and receiving data via TCP/IP over a network connection to a Terminal Server system. One example of a thin client is an Intel 386 processor running Windows for Workgroups 3.11 with a VGA graphics card and a network adapter. Third-party hardware vendors are releasing new windows-based terminals that satisfy Terminal Server's thin-client criteria with Windows CE, and MetaFrame lets X terminals, Macintosh computers, PCs running MS-DOS, Web browsers, and UNIX-based systems connect to Terminal Server.

Unlike other thin-client solutions, including X terminals, Terminal Server does not store state information on its clients or let clients participate in any part of application execution. Instead, Terminal Server feeds drawing orders (e.g., lines, fills, text drawing) and incremental changes of the computer's desktop. The only output processing Terminal Server clients must perform is reading the drawing commands (or presentation data) from the network connection and displaying the results on the screen. The clients' input processing consists of sending keystrokes and mouse-movement packets to Terminal Server. The advantage of this approach is that a user can move from one client system to another while maintaining the same connection state. The disadvantage is that the server must do most of the processing; therefore, resources such as CPU horsepower and memory size must be concentrated on the server.

The default communications protocol between Terminal Server and its clients is called Remote Desktop Protocol. RDP is based on International Telecommunications Union's (ITU's) T.120 protocol. The T.120 protocol is an international-standard conferencing protocol that Microsoft uses in its NetMeeting remote-conferencing product. The T.120 protocol supports 64,000 communication channels over the same connection so that data can divide into logical streams: for instance, keyboard input and video drawing orders. The data format of the channel in the first release of Terminal Server is based on the T.128 protocol. Terminal Server's channel uses only one channel connection that the keyboard, mouse, and presentation data share.

Terminal Server supports many plug-in protocols (e.g., Citrix's ICA protocol). Terminal Server adds these protocols to RDP when the Terminal Server add-on products load. In addition, although Terminal Server currently supports only RDP over TCP/IP, Terminal Server ships with drivers for other network protocols, including NetBEUI and IPX/SPX. Citrix's MetaFrame includes the necessary components to enable these other protocols, and Microsoft has indicated that Terminal Server might enable them in future releases.

A user who operates a typical Terminal Server client machine to connect to a Terminal Server system sees a configuration that is virtually identical to the console of a computer running NT. The illusion of an NT console is so thorough that, unless you enter autologon parameters into the Terminal Server client software, the user sees a logon screen. Microsoft designed NT under the assumption that only one user, the one operating the console, will use a graphical desktop. Modifying NT's architecture is necessary for Terminal Server to provide multiple interactive sessions for multiple users.

The one-graphical-user assumption most affects two parts of NT's architecture: Win32 and the Object Manager. The Win32 subsystem manages NT's GUI environment. The standard version of NT has one graphical logon session, which includes one keyboard and one mouse. Win32 takes mouse and keyboard input and funnels it to the appropriate application. The application responds to the input by invoking Win32 drawing and windowing commands to change the state of the application's windows and display information inside the windows. However, Terminal Server can manage multiple input and output devices. Furthermore, the graphical environments of each logon session must be totally independent; that is, the windows in one Terminal Server session are not visible in another session, and multiple foreground windows (one for each session) and multiple notions of a current mouse position and keyboard state can exist.

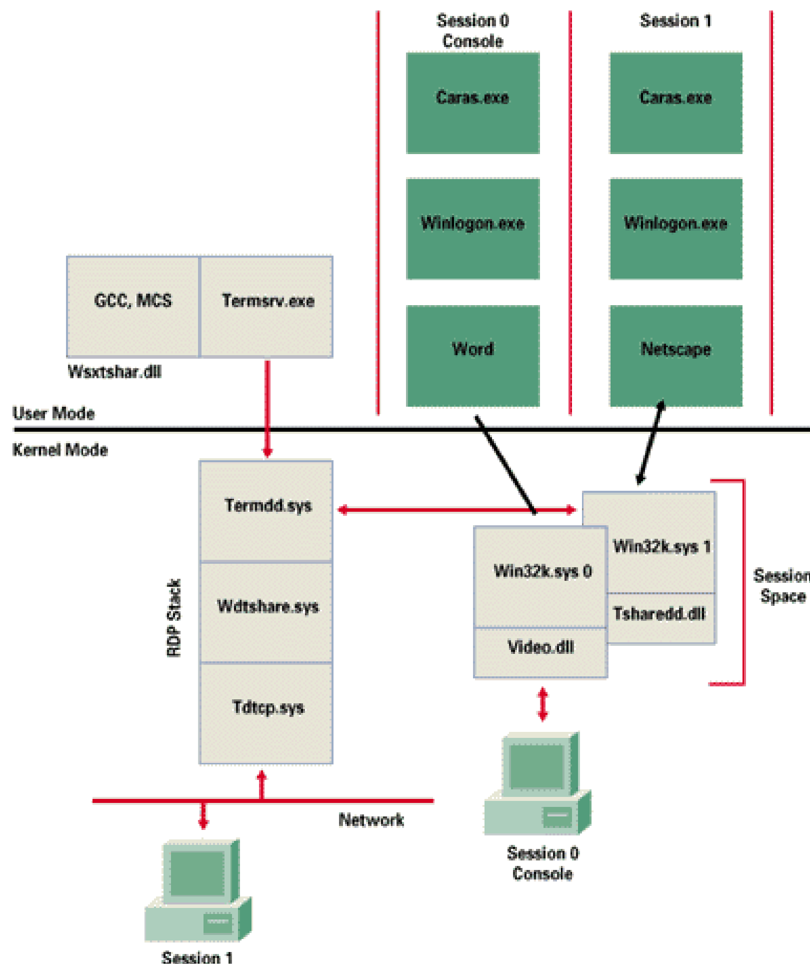
With the one graphical display that the standard version of NT implements, only one user can run a graphical application at a time. In addition, many standard NT applications prevent users from invoking multiple instances of the application because the application developers assumed the applications will be installed on workstations dedicated to a particular user. Complex programs sometimes consist of separate processes that use named objects to communicate and synchronize with one another. (For a complete description of the Object Manager and named objects, see my October 1997 NT Internals column.) For example, consider an application that consists of two processes, one of which must notify the other when a particular action takes place. The first process might create an event object called AppEvent that the other process can locate by name. A thread in the second process can reference the event and pass an identifier for it to an API, WaitForSingleObject, that will suspend a thread until a thread in the first process signals the event.

If more than one instance of an application could run in NT's standard version, the application's second instance could not create AppEvent, because AppEvent would already exist in the application's first instance. If the application did not detect the discrepancies between its first and second instances and abort, problems would occur because the second process of the second

instance would wait for an action the first process of the first instance signals. This scenario points to the kinds of problems that arise because the Object Manager namespace is global and developers write applications for use by only one logged-on user at a time. In a multiuser environment, allowing for multiple instances of word-processing, database client, and spreadsheet applications is desirable.

The Terminal Server Service

You can see Terminal Server's basic architecture in Figure 1. The control center is in the Terminal Server service, which NT implements in the <winnt>\system32\termsrv.exe file. The Terminal Server service manages client connections and starts when the system boots. The Terminal Server service is similar to other privileged systems, such as Win32, in that it is always running. The service keeps track of which clients have active and inactive connections and initiates the creation and shutdown of connection contexts. Terminal Server considers client connections to be logon sessions and assigns each client connection a unique session identifier called a Session ID. The Terminal Server service treats each console logon session as a client connection but reserves Session ID 0 for it. Thus, when you connect to Terminal Server, the Terminal Server service might assign Session ID 5 to your session. If you then log out and log back in from the same client, the Terminal Server service will assign a different Session ID to your new session. (Applications are oblivious to this process because it is a convention Terminal Server uses to track logon sessions.) The Terminal Server service also provides functions for enumerating, disconnecting, and querying information about logon sessions. The Terminal Server Administration application uses these APIs to present a control interface to the logon sessions.



Internally, NT calls logon sessions WinStations (NT 5.0 will call WinStations Session objects), because each logon session has an associated WinStation object. When a user initiates a new logon session, NT creates a new WinStation object, which stores permissions that use administrative tools to protect manipulations of the session. Although the name WinStation is similar to Windows Station, WinStations are not related to Windows Stations. The Win32K subsystem in kernel mode uses Windows Stations to track graphical logons. You can associate a graphical logon with one or more Desktop objects that represent distinct graphical desktops. (You can create and alternate multiple desktops with utilities such as the Microsoft Windows NT Server Resource Kit's Virtual Desktops.)

To provide fast connections to Terminal Server, Termsrv (Termsrv.exe in Figure 1) always ensures that two idle WinStations exist. Termsrv calls the Session Manager process (<winnt>\system32\smss.exe) to create a new WinStation. Session Manager attaches itself to the new WinStation. From that point forward, Session Manager associates all processes it creates with the new logon session. However, before it creates any new processes, Session Manager loads a new instance of Win32K, the Win32 kernel-mode subsystem. Next, Session Manager creates a Winlogon (<winnt>\system32\winlogon.exe) process, and then an instance of the Win32 Client-Server Runtime Subsystem (CSRSS, located in <winnt>\system32\csrss.exe). Winlogon is the first client of a CSRSS instance and is charged with logon authentication, and CSRSS is the process-and-thread manager of a logon session. Only after Termsrv assigns the session that consists of these processes to a client connection are the processes fully running. After Termsrv attaches the session to a client, the processes display the Begin Logon prompt.

A new client connection results in the client's association with an idle WinStation. When a user logs on, Winlogon calls CSRSS to create the logon session's graphical shell (typically Explorer). Because Terminal Server associates CSRSS with a WinStation, every process CSRSS creates inherits the same association. This association tags every process with a Session ID. In addition, each instance of CSRSS is aware of only the processes it creates in its own session. Processes that CSRSS creates in one session speak only to the CSRSS that created them because they use a per-WinStation API communications port to talk to the CSRSS.

After a logon takes place, Winlogon associates the logon session's Session ID with the domain and account name of the user who logged on. Winlogon then queries Termsrv to see if the user has an existing disconnected session. If Winlogon doesn't find a disconnected session, it launches the shell. If Winlogon finds one disconnected session, it notifies Termsrv to transfer the client's connection to the disconnected session. This action reconnects the client to the session it left open and sends the session it was tentatively associated with back to the idle state. If Winlogon finds more than one disconnected session, it displays these sessions in a list and lets the client user select the session to reconnect to. This powerful feature lets a user roam among client machines while maintaining one logon session on the server--a useful capability when a client system fails because of a hardware problem.

Another interesting RDP feature is that you can reconnect to existing sessions at different resolutions. For example, if you connect to Terminal Server at a resolution of 1024 * 768 and disconnect, you can reconnect at 640 * 480. RDP detects the resolution differences and switches video modes on the reconnected session. This process is similar to what happens when you dynamically change video resolutions in standard NT 4.0.

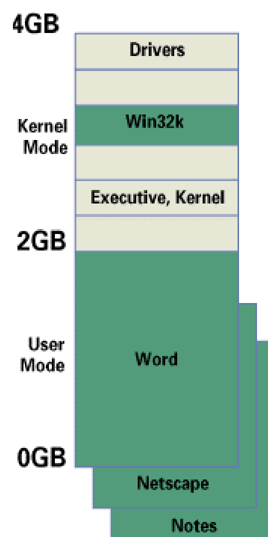
The Session Space

The other components of a logon session that Figure 1 shows include the Win32 kernel-mode subsystem (win32K.sys) and a display driver. A Terminal Server console session's display driver is a standard display driver a video card vendor supplies. The display driver provides an interface for the video card's capabilities and hardware so that Win32 can display graphics. Identical video drivers ship in standard versions of NT. A Win32K instance for a Terminal Server client connection cannot use the console's display but must instead send drawing orders to the client machine. Thus, Terminal Server provides a virtual display driver (tsharedd.dll) for nonconsole sessions. This virtual display driver appears to Win32K exactly like a real hardware video driver, but instead of writing to the video card's graphics memory, the virtual display driver writes into buffers that represent the client's display. Terminal Server's communications stack transmits the buffers to the clients.

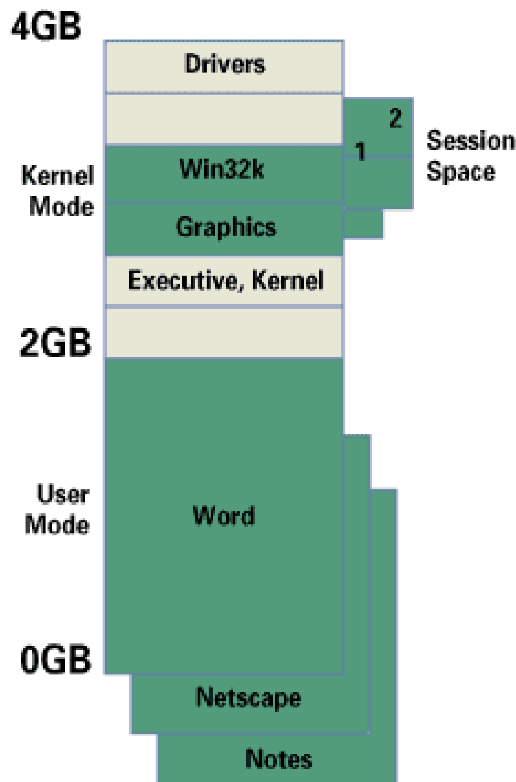
Terminal Server's designers had two design choices for making the Win32K subsystem handle multiple logon sessions. The first choice was to have one instance of Win32K that knows about all logon sessions and maintains state information specific to each session (such as a list of windows, the windows' positions, cursor position, and all other data the standard version of NT's Win32K keeps track of for a graphical logon). This approach would require extensive modification to Win32K's internal data structures and control routines, because Microsoft wrote Win32K under the assumption that only one set of input devices and one output device exist.

The second design choice, which Citrix used with WinFrame and the Terminal Server design team used, was to create a new instance of Win32K for each logon session. The advantage of this design is that it requires minimal changes to Win32K, so Win32K still assumes there is one set of input devices and one output device. However, if you're familiar with NT's virtual memory architecture, you know that a problem had to be overcome before the second design choice would work.

The standard NT memory map, which Figure 2 shows, comprises one 4GB region divided into two 2GB parts (in NT 4.0, Enterprise Edition, you can opt to reconfigure this division to 3GB and 1GB). The lower 2GB portion contains application-specific mappings, and this portion changes to reflect the application currently executing on the CPU. The upper 2GB portion is fixed and contains the kernel-mode portions of NT, including device drivers, the hardware abstraction layer (HAL), the Kernel, and the Executive subsystems. The upper memory portion does not recognize multiple address spaces; however, supporting multiple instances of Win32K requires recognition of multiple address spaces.



To overcome this problem in the upper memory portion of NT's memory map, Terminal Server's memory management extends NT's basic model to support one region of memory in the kernel-mode half of the memory map. This region is called the Session Space, and it can switch among different mappings. The Session Space occupies a fixed 80MB range of the virtual memory map just above where the Kernel and other Executive subsystems reside, and NT places different instances of Win32K in this range. The RDP graphics driver also instantiates in this area. When Terminal Server creates a new session, Session Manager uses a new system call to direct the mapping of Win32K, the RDP display driver, and kernel-mode print drivers into the Session Space, as Figure 3 shows.



NT uses the same mapping mechanism in the Session Space that it uses to map applications in the user-mode part of virtual memory. NT initially marks the entire Session Space as invalid for a logon session. However, as Win32K and RDP execute in the Session Space, NT's Virtual Memory Manager retrieves the appropriate code or data from the files (in this example, win32k.sys or rdpdd.dll) and makes them valid. NT performs an important optimization: Memory mappings of the Session Spaces of all the logon sessions share the physical memory that stores the nonchanging parts of the code. This optimization mirrors the way NT shares, in the user part of virtual memory, code from system DLLs that map simultaneously into different processes. Section objects represent memory that two or more processes share. Terminal Server also virtualizes these objects, so only those processes that belong to the same session can share the memory.

The Rdpdd display driver is known as the RDP display driver. Win32K sends graphics output commands to the RDP display driver. Rdpdd processes the drawing command and sends it to the client via the RDP protocol stack (wdrdp.sys). Win32K instances associated with remote clients also receive mouse and keyboard input via Rdpdd, whereas the console instance of Win32K receives mouse and keyboard input from hardware on the console machine. Rdpdd's job is to pass output to,

and obtain data from, communications endpoints for multiple client sessions. Terminal Server creates the endpoints in the wdrdp.sys driver, which implements, with the aid of rdpdd, the RDP protocol. Wdrdp sits on top of one or more Transport Stack Drivers (TDs). TDs talk to Terminal Server transport drivers that communicate using a particular networking protocol, such as TCP/IP, NetBEUI, or IPX/SPX. As I stated earlier, Terminal Server uses only the TCP/IP TDI driver (tdtcp.sys), but other drivers are present for MetaFrame to use (e.g., tdipx.sys, tdspx.sys, tdnetnb.sys).

The network-protocol layering I just described creates a stack of drivers called an RDP Listener Stack. A client first connects to Terminal Server through the RDP Listener Stack. Terminal Server then transfers the connection to a new instance of an RDP stack, and the Listener Stack returns to the listening state. When MetaFrame extensions load, an ICA protocol driver loads into the stack alongside Wdrdp. Termsrv also plays a role in communications with the client and the management of client connections by loading a WinStation Extension DLL. The extension DLL for the RDP protocol is wsxrdp.dll, and in this DLL the T.120 General Conferencing Control (GCC) component creates T.120 conferences and manages channels for those conferences with the aid of the T.120 Multipoint Communications Service (MCS) component. The GCC and MCS components are both part of the T.120 standard. MCS's conference-related management routines are in user mode in the Wsxrdp extension, whereas its critical data paths (such as writing data to a conference) are in kernel mode in wdrdp.sys. Terminal Server implements this split so that RDP's video display driver (rdpdd.dll) will have direct access to the RDP protocol stack (wdrdp.sys) without having to leave kernel mode. Switching between kernel mode and user mode would introduce a performance penalty.

Object Management

The Object Manager in Terminal Server is called the Multi-User Object Manager, because it knows how to instantiate parts of the Object Manager namespace. When an application creates a named object (such as AppEvent from my earlier example), the Multi-User Object Manager determines whether the object's name should be global across all logon sessions or specific to a particular session. Table 1 lists all the object types in the standard version of NT. If you examine the list, you'll realize that objects of certain object types are global to the system, whereas other objects are local to the session that creates them. For example, files and Registry keys should be global, but synchronization events (such as AppEvent) and other synchronization objects (mutants, timers, and event pairs) should be local. Therefore, by default, the Multi-User Object manager localizes all objects of the types Table 2 shows, and globalizes the rest. Special cases can arise in which this heuristic approach would break an application; thus, Terminal Server has a set of application compatibility flags that it sets to override this behavior for certain programs.

TABLE 1: Object Types in NT 4.0

Object Type	Represents	Defining Subsystem
Object type	Object type object	Object Manager
Directory	Object namespace	Object Manager
SymbolicLink	Object namespace	Object Manager
Event	Synchronization primitive	Executive
EventPair	Synchronization primitive	Executive
Mutant	Synchronization primitive	Executive
Semaphore	Synchronization primitive	Executive
Timer	Timer notifications	Executive

Windows Station	Logon session	Win32
Desktop	Windows desktop	Win32
File	Tracks open files	I/O Manager
I/OCompletion	Tracks I/O completion notifications	I/O Manager
Adapter	Direct Memory Access (DMA) resource	I/O Manager
Controller	DMA controller	I/O Manager
Device	Logical or physical device	I/O Manager
Driver	Device driver	I/O Manager
Key	Doorway to the Registry	Configuration Manager
Port	Communications channel	Local Procedure Call (LPC) Facility
Section	Memory mapping	Memory Manager
Process	Active process	Process Manager
Thread	Active thread	Process Manager
Token	Process security profile	Process Manager
Profile	Performance monitoring	Kernel

TABLE 2: Localized Object Types in Terminal Server
Directory
SymbolicLink
Event
EventPair
Mutant
Semaphore
Timer
Windows Station
Desktop
Port
Section

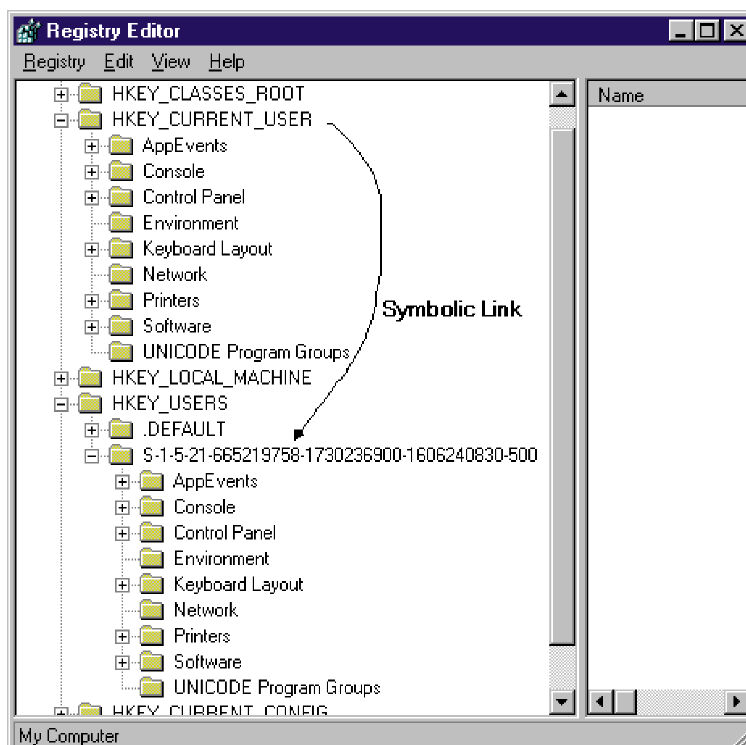
Terminal Server checks an object's type when an application creates the object, and if the object's type should be private to a session (or user-global), Terminal Server tags the object with the session's ID. If the object is not user-global, Terminal Server identifies the object as system-global. An object's Session ID acts like an extension to the object's name. When an application creates a new object, the new object will collide with an existing object only if another object of the same name exists within the namespace the application references. For example, if the first instance of an application creates AppEvent in the first instance's session (i.e., the object is user global) Terminal Server will tag that AppEvent with the application's Session ID. When the application's second instance creates its AppEvent, it will succeed because its Session ID is different from the Session ID of the first instance of AppEvent; hence, the second object will appear in the namespace of the second AppEvent's session. Only global objects and objects whose Session ID matches an application's Session ID will be visible to the application--the Object Manager will hide all other objects from the application. This process will change slightly in NT 5.0, which creates a separate namespace directory for each session. Session IDs will identify the namespace directories NT 5.0 creates. In addition, NT 5.0 places

objects it creates with a Global\ prefix in the global namespace, objects it creates with a Local\ prefix in the session's (local) namespace, and objects it creates with a Session\<sessionid> prefix in the specified session's namespace.

Terminal Server hasn't modified standard NT's thread-scheduling algorithms. Terminal Server's thread scheduler is identical to standard NT's thread scheduler, which means that Terminal Server's thread scheduler is unaware of sessions. The session a thread belongs to does not influence the scheduler's decisions.

Administrative and Application Issues

You might wonder how Terminal Server relates to the use of the Registry. Applications typically store configuration information in one or more of the following places: in files (such as .ini files), under HKEY_LOCAL_MACHINE\SOFTWARE in the Registry, or under HKEY_CURRENT_USER\Software in the Registry. Since NT's first release, Microsoft's guidelines have been to store system-global settings (such as the location of application files) under HKEY_LOCAL_MACHINE\SOFTWARE, and user-specific settings (such as background color preferences) under HKEY_CURRENT_USER\Software. It is especially important that applications that run in several logon sessions adhere to these guidelines. If user-specific information is stored in files or under HKEY_LOCAL_MACHINE\SOFTWARE, collisions between different users will occur. HKEY_CURRENT_USER is safe for user-specific settings, because the key is a symbolic link that points to the key that belongs to a user under HKEY_USERS, as Screen 1 shows. Terminal Server supports different links for different logon sessions so that users will see their data under HKEY_CURRENT_USER. In addition, Terminal Server contains a large amount of application-compatibility code to tolerate applications that don't adhere to these guidelines. This code makes it possible for Terminal Server to run these programs from different sessions. However, this support is provided only for existing applications that were written without knowledge of multiuser problems--new applications must follow Terminal Server's rule to work properly.



Another way in which Terminal Server differs from standard NT relates to administration. In Terminal Server, administrators can use the Application Restriction administration tool to specify that clients can execute only certain applications. Terminal Server maintains a list containing these clients in the Registry at:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TerminalServer\Authorized Applications\Application List

When an administrator uses the Application Restriction tool to change the list, a watchdog in the Kernel notices and updates its internal table of allowed applications. Whenever an application executes, NT calls a function that checks the application's path against the paths in the Authorized Applications list. If there isn't a match, Terminal Server denies the attempt at execution. (Administrators are exempt from this control.) The default behavior is to let any application execute. The Application Restriction tool is meant for administrators to use primarily when Terminal Server is serving anonymous connections to the Internet. The tool, in combination with the Zero Administration toolkit that runs on Terminal Server, gives administrators substantial control over their users' working environments.

The final administrative feature I'll describe is Terminal Server's support for secure communications in Wdtshare. There are three possible security settings: low (the default setting), medium, and high. In low security, Terminal Server encrypts packets a client sends to the server with the Microsoft-RC4 algorithm. Low security protects a client's password and other sensitive information (such as a credit card number) as it travels to the server. In medium security, Terminal Server encrypts packets traveling from either client to server or server to client with Microsoft-RC4 so that the client's display is safe from network snoopers. Finally, in high security, Terminal Server encrypts packets traveling from client to server and server to client but uses the industry-standard RC4 algorithm. All three encryption schemes use 40-bit keys, except for the high-security mode of the US-only version of Terminal Server, which uses 128-bit keys.

Expanding NT

Terminal Server is a major addition to the NT product line and will help to broaden NT's acceptance. With Terminal Server's modifications to system address space memory management, support for object namespace instantiating, and extensions to Win32K (all of which let NT interact with different input and output devices), NT can serve CPU cycles and applications as well as files and printers.