Windows Operating System Services, Functions, Routines, Processes, Threads, and Jobs

Services, Functions, and Routines

Several terms in the Windows user and programming documentation have different meanings in different contexts. For example, the word *service* can refer to a callable routine in the operating system, a device driver, or a server process. The following describes what certain terms mean in this book:

• **Windows API functions** Documented, callable subroutines in the Windows API. Examples include *CreateProcess. CreateFile, and GetMessage*.

• **Native system services (or system calls)** The undocumented, underlying services in the operating system that are callable from user mode. For example, *NtCreateUserProcess* internal system service the Windows *CreateProcess* function calls to create a new process.

• **Kernel support functions (or routines)** Subroutines inside the Windows operating system that can be called only from kernel mode (defined later in this chapter). For example, *ExAllocatePoolWithTag* is the routine that device drivers call to allocate memory from the Windows system heaps (called pools).

• **Windows services** Processes started by the Windows service control manager. For example the Task Scheduler service runs in a user-mode process that supports the *at* command (which is similar to the UNIX commands *at* or *cron*). (Note: although the registry defines Windows device drivers as "services," they are not referred to as such in this book.)

• **DLLs (dynamic-link libraries)** A set of callable subroutines linked together as a binary file that can be dynamically loaded by applications that use the subroutines. Examples include Msvcrt.dll (the C run-time library) and Kernel32.dll (on of Windows API subsystem libraries). Windows user-mode components and applications use DLLs extensively. The advantage DLLs provide over static libraries is that applications can share DLLs, and Windows ensures that there is only one in-memory copy of a DLL's code among tha applications that are referencing it. Note that nonexecutable .NET assemblies are compiled as DLLs but without any exported subroutines. Instead, the CLR parses compiled metadata to access the corresponding types and members.

Processes, Threads, and Jobs

Although programs and processes appear similar on the surface, they are fundamentally different A *program* is a static sequence of instructions, whereas a *process* is a container for a set of resources used when executing the instance of the program. At the highest level of abstraction, a Windows process comprises the following:

• A *private virtual address space*, which is a set of virtual memory addresses that the process can use

• An executable program, which defines initial code and data and is mapped into the process' virtual address space

• A list of open handles to various system resources - such as semaphores, communication ports, and files - that are accessible to all threads in the process

• A security context called an *accesstoken* that identifies the user, security groups, privileges, User Account Control (UAC) virtualization state, session, and limited use account state associated with the process

• A unique identifier called a *process ID* (internally part of an identifier called a *client ID*)

• At least one thread of execution (although an "empty" process is possible, it is not useful)

Each process also points to its parent or creator process. If the parent no longer exists, this information is not updated. Therefore, it is possible for a process to refer to a nonexistent parent. This is not a problem, because nothing relies on this information being kept current. In the case of ProcessExplorer, the start time of the parent process is taken into account to avoid attaching a child process based on a reused process ID.

The above is an excerpt from: