Event Tracing

General Overview

Chapter 13 of Developing WMI Solutions covers a high performance instrumentation technology within Windows called *Event Tracing*. Due to some of the questions I've had recently, it has occurred to me that I could better explain some of the practical uses of this technology. Let's start by covering some of the same ground as the introduction of Chapter 13.

The computing world has long required the instrumentation of applications with detailed trace information and techniques to monitor system usage. The most basic trace instrumentation that many development organizations employ is a debug logging facility. Debug logging is usually achieved by writing text statements into a text file that may include the time, a thread identifier, and the subsystem or component writing the trace statement. This basic form of instrumentation has aided many organizations to resolve problems by inspecting the log file to determine the program flow at the time a problem occurred. The log file has other benefits apart from identifying where a particular problem occurred. A developer can use the log file for performance profiling by inspecting the tick count (time) and identify portions of code that take too long to complete. A developer can identify multithreaded problems by seeing the sequence of calls made between threads and components. Even if a problem cannot be easily reproducible, a developer can ascertain from the log file where additional instrumentation should be added and make the section of code more robust.

Every line or statement written to a debug log file can be viewed as an event, hence the name of a new instrumentation technology introduced in Windows 2000, event tracing. Event tracing is a subsystem that is deeply integrated in the Windows operating system and is considered part of the Windows Management Instrumentation (WMI) tool set. Event tracing is super fast and provides new scope for resolving problems and monitoring and tracking resources. The most fundamental aspect of event tracing is an event. An event can be best described as an activity of interest. For example, the Windows operating system's TCP/IP stack is instrumented with event traces that describe activities such as connect, send, receive, and disconnect. Each event typically includes additional information about the activity: In the case of the TCP/IP stack, more information is provided about the connection and details of what was sent or received. The subsystem that implements event tracing is referred to as the event tracer and is implemented in the Windows kernel.

You may be familiar with another high performance instrumentation mechanism called *performance counters*. Since the release of Windows NT 3.1, all applications have been able to read and write high performance counters. High performance counters offer a very fast solution for monitoring system utilization. This form of instrumentation is different from event tracing because the information communicated is based on a single value for a single counter at that moment. Performance counters are arranged into objects that can contain one or more counters and each counter in turn may have one or more instances. For example, a particular *Process* object has some counters like *Thread Count* for which there are several *Instances* (that is, currently running applications). This is one example of the instrumentation provided by the operating system. In fact, Windows includes many performance counters: Other examples include Memory, IP, Processor, and System. To make such an instrumentation technique useful, a monitoring application continually must retrieve the counter values in a loop that allows an ongoing graph or log to be updated. For example, the performance monitor supplied with Windows is a graphical application that can display graphs for one or more performance counters. A developer who wishes to identify whether an application leaks memory over a long period can set up the performance monitor to monitor the *Private Bytes* counter from the *Process* object. If the

graph goes up, then the application is leaking memory. If the graph is stable, then the application is well behaved. This is one example that demonstrates how tremendously useful software instrumentation is. It is also an example of how non-instrumented applications can benefit from the instrumentation provided through the Windows operating system. Many server-side applications now instrument or expose high performance counters to help administrators in planning for system capacity. Writing code to access performance counters is not user-friendly and requires the use of the registry APIs. To overcome the difficulties of accessing the performance counters, Microsoft developed the Performance Data Helper (PDH) library. This makes it significantly easier to write code to monitor performance counters.

The main difference between event tracing and performance counters is that it is hard to track individual resources with performance counters. Performance counters offer a way to view system utilization, like memory usage. Performance counters cannot be used easily for debug logging, for tracking resources or operations such as a disk drive, or for monitoring registry access.

Event tracing is based on a publish and subscription mechanism. That is, the events are published once and the event stream can be subscribed to many times (i.e. by many applications or threads). This mechanism makes event tracing a suitable means to develop various utilities and monitoring applications which can all get access to the same published event stream.

Let's have a look at Figure 1. It shows an event tracing provider publishing events to the operating system. There are five applications each receiving the event stream and performing useful tasks either with the information or storing it in an alternative format. This is one of the key benefits of event tracing. It leaves the door open for you to develop various monitoring tools that check for system health, system capacity, operational analysis and resource tracking. The operating system can also store the event stream to a file using one of many possible configurations. For example, you could specify that the event tracing log file be circular and limited to 10MB in size. Letting the operating system save the event stream to a file is another key benefit. It enables you to develop various post analysis tools which may produce a summary, or store the data into an alternative searchable format, or look for trends, or simply view the events in their native format (such is the case of the Event Tracing Walker application, Figure 2, which is also available from the website).

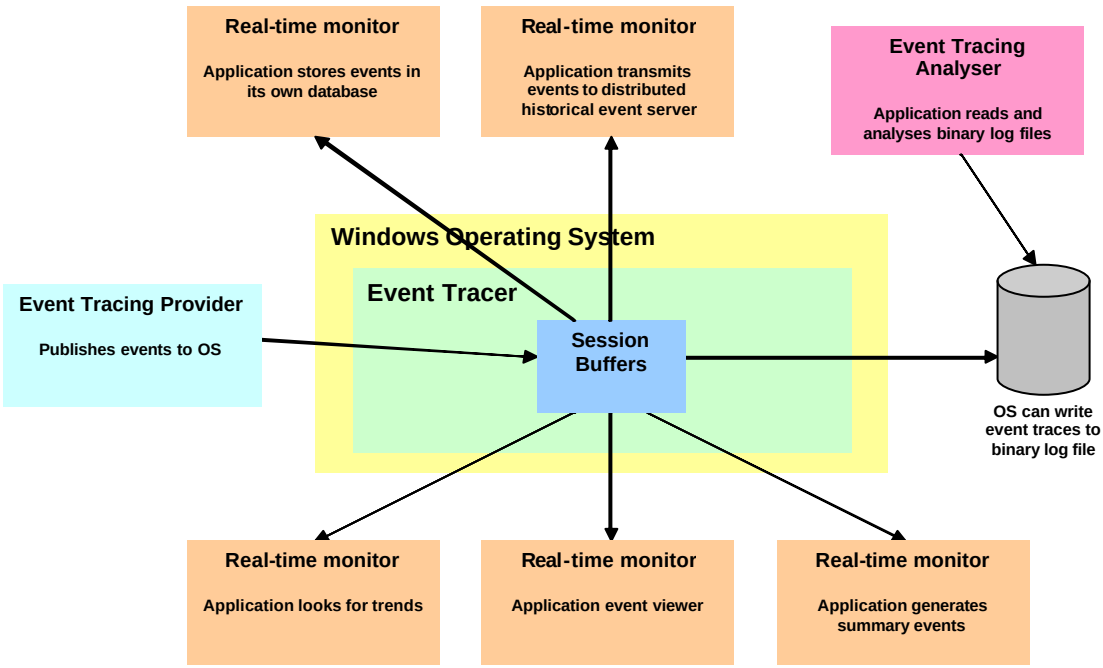

Figure 1: View of published events being delivered to subscription applications

Every event is a binary structure that has some predefined fields, such as a time stamp, the type of event, and its version. Most events are extended with customized fields or structures to supply

additional information about the activity. The Windows kernel exposes many event traces which include activity with processes, threads, file and disk activity, TCP/IP, loading images such as DLLs and executables, registry access, and hard and soft page faults. There are also event traces for the various security subsystems used by Windows. Windows XP and Windows Server 2003 extends the existing kernel events with new versions to provide even more detail about what is happening in the kernel.

Once your software services or components publish event traces, it becomes possible to get several event streams from the system (including the kernel). The event tracing technology allows you to analyse several event tracing log files at the same time and the operating system can provide you with a time-ordered event stream. This facility can also make it possible to resolve distributed problems where event tracing logs can be obtained from each server. There is much scope here to match events around the same time across the affected servers for a particular operation or resource.

Event tracing doesn't stop there! A single event in an event trace can have a parent event. This opens up the possibility to have child events that detail a larger event. Suppose an application has to perform a complex calculation. It could fire a start calculation event followed by child events that detail the progress of the calculation which might finally end in a stop calculation event. So, not only can you drill down through all the components to the operating system, but you may also be able to drill through all the child events and their child events. This cannot easily be achieved with many other event streaming mechanisms such as a flat text file.

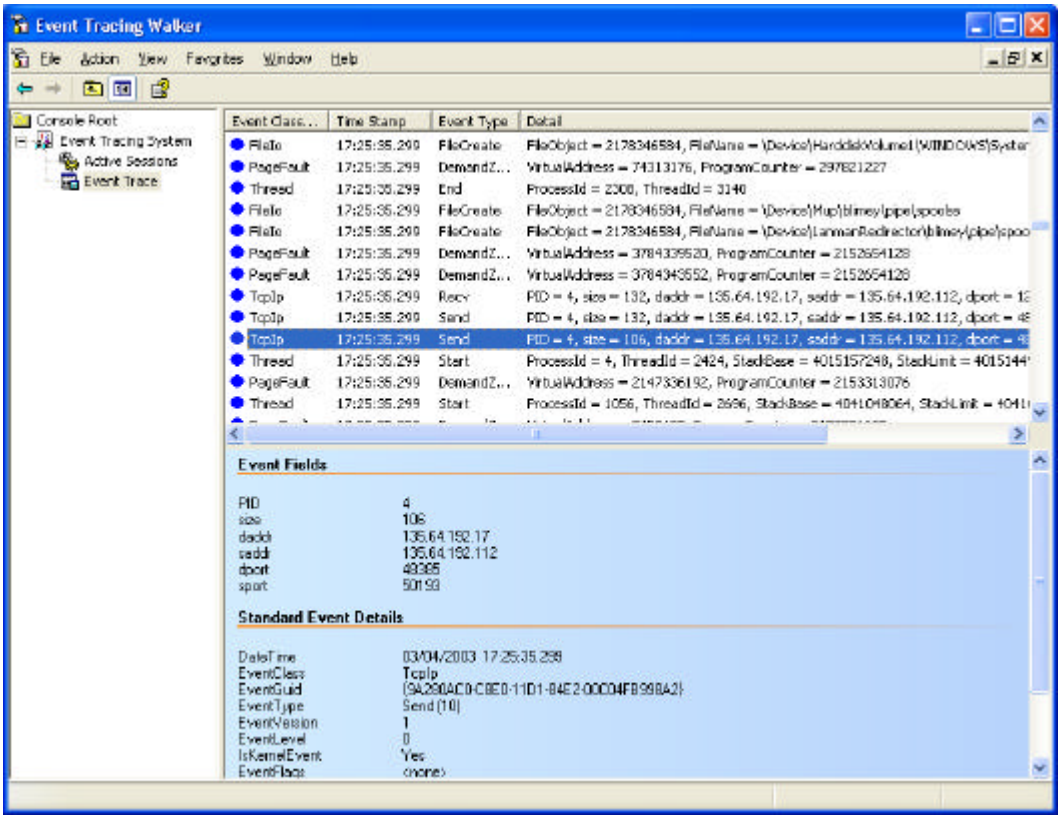

Figure 2: The Event Tracing Walker showing some kernel events

The operating system's event tracer temporarily holds the events published by a provider in non-paged system memory buffers. The event tracer manages the buffers and if necessary, it also manages the dumping of them to an event trace log file on disk. The event tracing APIs are super fast, so any time-sensitive operation should not be affected by using them. In fact, on multiprocessor systems, each processor is allocated a separate active buffer to eliminate contention.

A limitation of the event tracing system is that events delivered to monitoring and analysis applications may not appear in the exact sequence that the events were delivered to the event tracer. This is true only if there are multiple events within the same millisecond. This may alter the type of events you publish where an event is not necessarily associated with the previous or next event.

Event tracing paves the way for unified debug, resource, operation, and capacity planning activities that have not been previously possible, especially when the kernel, device drivers, and security activity are included.

Example case study

Problem: Develop a distributed event tracing system.

Solution: One approach in solving this problem would be to use the Windows event tracing subsystem and the Microsoft® Message Queue Server (MSMQ). The event tracing system is super fast and allows you to further develop monitoring tools outside the scope of the initial problem domain. MSMQ provides the technology for reliable network asynchronous communication.

The basic operation involves each server having a Windows service which grabs the event traces locally and batches them up in MSMQ messages to be sent to a historical tracing server. The batching up of information would help to reduce network traffic. And when parts of the distributed system go down or network traffic from the server is interrupted, no trace information is lost which makes for a reliable and successful distributed tracing system. The historical tracing server on the network grabs the MSMQ messages pushed to it from the other servers and stores the event trace information in a database. Figure 3 shows an outline of the architecture.

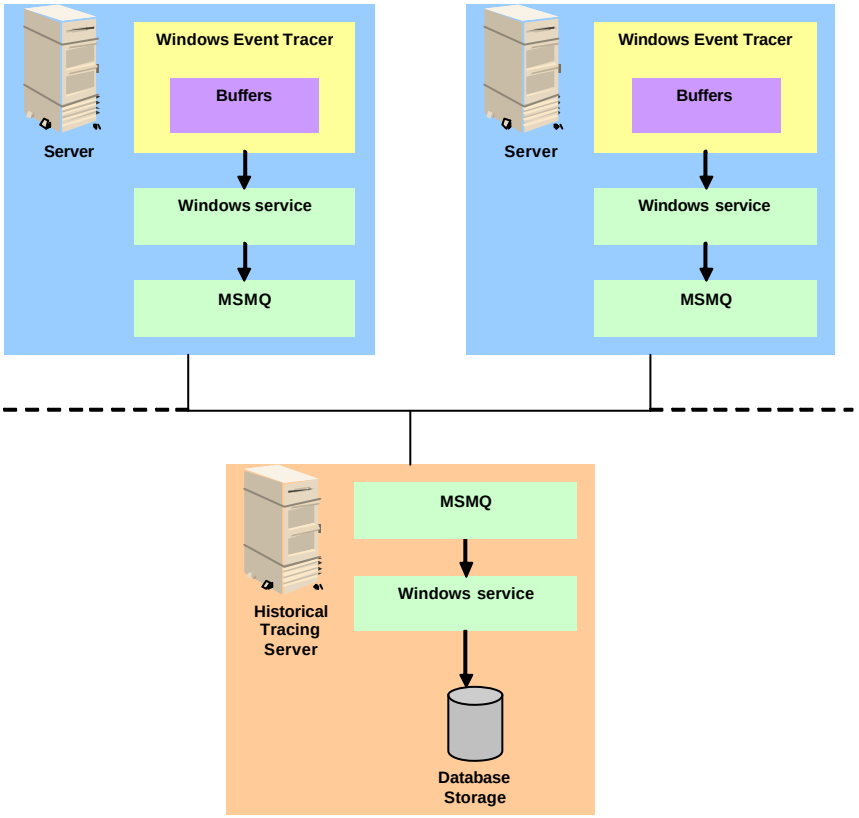

Figure 3: Each Server sends MSMQ messages to Historical Tracing Server