Software Tracing in Windows Drivers

들어가면서...

software tracing은 실행 중인 드라이버의 behavior를 아주 효율적인 방법으로 조사할 수 있는 기법이다. 물론, checked build나 debug print statements를 구현하기 위한 overhead는 당연히 없다. 이 번의 기사에서 다루는 내용은 Kernel Mode에 한하며 다음의 O/S들에 대해서 적용된다.

Codenamed “Longhorn”
Server 2003
Windows XP
Windows 2000

본문

software tracing 기법이 어떻게 구현되는지를 알아보기 전에 먼저 드라이버에서 software tracing기법을 사용하면 어떤 장점이 있는지 간단하게 몇 가지를 살펴보자.

Available in shipped products

우리가 원하는 trace log를 free build로 구현하여도 충분히 capture할 수 있다. 예전의 방식처럼 Kernel debugger의 도움을 받거나 checked build로 구현할 필요가 없다.

Low impact on performance

드라이버들은 trace message를 tracing이 enable되었을 때만 만든다. 또한 이 trace message들은 binary 형태로 생성되며 application에 의해 user가 사용하기 편한 형태로 변환된다. 따라서 tracing message를 생성하는데 많은 overhead가 필요하지 않다.

Dynamic and flexible

Tracing은 동적으로 enable/disable이 가능하며 Tracing에 관여하는

각종 module이 동작하는데 operating system의 rebooting이 필요 없다.

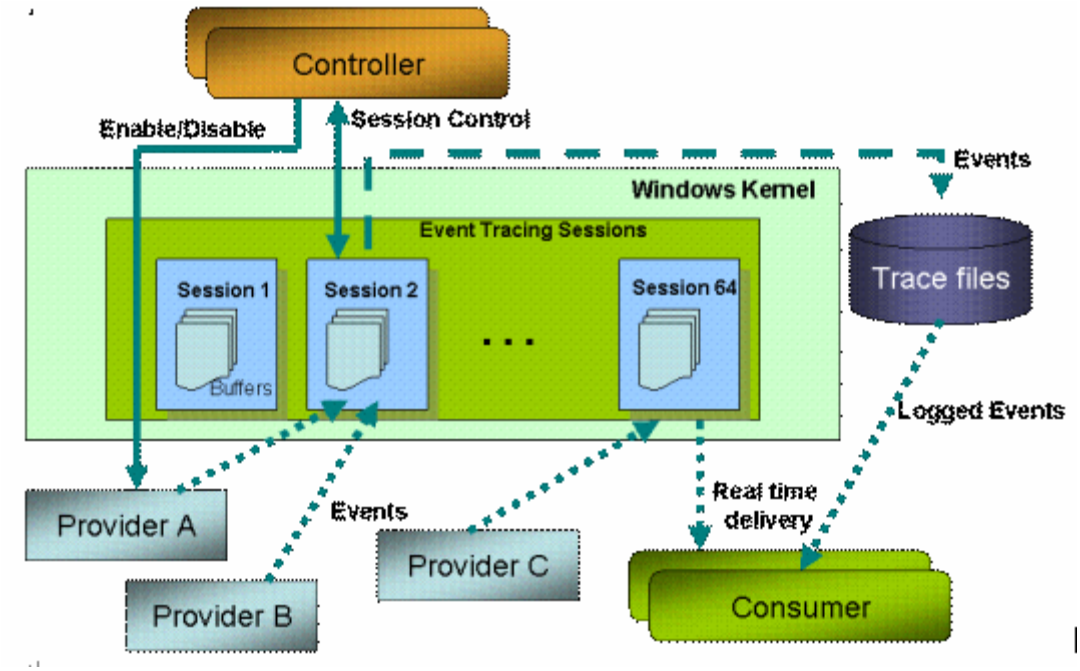
Rich trace information without exposing intellectual property

필요에 따라 trace message에 timestamp, function name, CPU 정보, Process, Thread정보 등의 추가적인 사항을 자동적으로 포함시켜 준다. 예전처럼 debug message에 일일이 필요한 모든 사항들은 직접 coding할 필요가 없어진 것이다. 이러한 것들은 ETW(Event Tracing for Windows)에 의해서 구현되어 진다.

Easy migration from debug print statements

WPP(Windows software trace preprocessor)는 드라이버에서 ETW(Event Tracing for Windows)를 쉽게 구현하게 해준다. 또한 WPP는 기존의 debug print function을 드라이버 build시 trace function call로 변환을 시켜주는 것이 가능하므로 기존의 code를 수정함이 없이 software tracing기능을 가능하게 해준다.

본격적으로 이제 어떻게 software tracing이 구현되는지를 알아보자.
먼저, <그림-1> 을 살펴보자.



<그림-1>

<그림-1>은 software tracing Architecture를 나타내고 있다. 각각의 구성요소의 역할에 대해서 간단하게 알아보자.

Trace Providers

Trace provider는 tracing기능을 갖추고 있는(즉, trace message를 발생할 수 있는 code를 포함하고 있음을 말한다) application, o/s 구성요소, 또는 드라이버이다. 그리고 이 trace providers은 trace controller에 의해 trace기능이 enable되어야만 message를 발생시킨다.

Trace Sessions

Trace session은 하나 또 여러 개의 trace providers가 하나의 event log에 trace messages를 발생시키는 기간(period)을 말한다.

Trace Controllers

Trace controller는 trace sessions을 관리하며 또한 trace message를 trace session buffer로부터 event trace log(ETL) file 또는 직접 trace consumer로 전달하는 기능을 담당한다. 여기서 trace consumer는 trace message를 전달 받아, formats을 거친 다음 display하는 기능을 담당하는 application또는 tool들이다. Trace consumer는 trace message를 사람들이 쉽게 읽을 수 있는 형태로 format을 하는데 이때, 참조하는 file들이 provider's PDB symbol file 또는 TMF file이다. 물론 이러한 file들은 providers이 build될 때 생성되는 file들이다.

이상으로 간단하게 software tracing architecture에 관련된 구성요소를 살펴보았다. 계속해서 이제는 드라이버는 어떤 기능을 갖추어야만 provider로서 역할을 할 수 있는지 알아보자. 먼저, ETW와 WPP에 대해서 간단하게 살펴보자.
사실 windows kernel-mode tracing은 ETW에 기반을 두고있다. 따라서 드라이버에서 software tracing 기능을 구현하기 위해서는 MOF(managed object format) file을 이용하여 WMI provider로서 등록을 하여야 하며 trace event에 대한 여러 가지 작업을 수행 후 event 발생 시 log를 위한 작업을 수행하여야 한다. 이러한 복잡한 작업을 WPP가 간단하게 구현을 가능하게 해준다.
드라이버는 WPP의 macro를 이용하면 드라이버 build시 WPP는 macro와 template를 이용하여 드라이버가 trace provider로 등록되게 하고 trace message를 발생시키게 하는 code를 만들어 준다.
자, 이제부터 앞서 언급한 내용을 기반으로 하여 실제 예제를 보면서 구체적으로 드라이버에서 어떻게 software tracing을 구현하는지를 6단계로 나누어

알아보자.
DDK “src₩general₩tracedrv” sample을 기반으로 하여 설명을 하기로 한다.

Step 1. Include the TMH file for each source code file that contains any WPP macros.

WPP macro를 사용하는 모든 driver source file은 tmh file을 include 하는데 WPP는 build시 source file과 이름이 같은 xxx.tmh file을 생성한다. Tracedrv.c를 보면 다음의 code를 볼 수 있다.

```
#include "tracedrv.tmh"         //   this is the file that will be auto generated
```

사실 tracedrv.tmh는 tracedrv를 build할 때 objchk_wxp_x86₩i386 folder에 생성된다. File을 open하여 보면 ETW에 연관된 작업을 위한 code과 생성되어져 있음을 볼 수 있을 것이다.

Step 2. Define a WPP_CONTROL_GUIDS macro that specifies a GUID and trace flag names for controlling tracing.

WPP_CONTROL_GUIDS macro는 trace provider를 위한 control GUID를 만들고 trace flag names를 정의한다. 이 control GUID는 ETW 와 WPP가 provider를 구별하는데 사용되어진다. 그리고 trace flag는 trace controller에 의해 enable되어질 수 있으며 provider는 flag를 기반으로 trace message를 생성한다.

```
//
// Software Tracing Definitions
//
#define WPP_CONTROL_GUIDS ₩
WPP_DEFINE_CONTROL_GUID(CtlGuid,(d58c126f, b309, 11d1, 969e, 0000f875a5bc),   ₩
        WPP_DEFINE_BIT(TRACELEVELONE)                    ₩
        WPP_DEFINE_BIT(TRACELEVELTWO) )
```

GUID는 GUIDGEN tool을 사용하면 된다. 그리고 control GUID와 friendly name은 driver’s control GUID file인 “tracedrv.ctl”에 다음과 같이 존재한다.

```
d58c126f-b309-11d1-969e-0000f875a5bc      CtlGuid
```

그리고 TRACELEVELONE, TRACELEVELTWO는 trace flag name로서 WPP는

trace flag의 해당 bit에 값을 할당한다. 자세한 bit 값들은 DDK inc folder에서 “evntrace.h”를 참조하기 바란다.

Step 3. Call WPP_INIT_TRACEING from the driver’s DriverEntry routine.

WPP_INIT_TRACING macro는 드라이버에서 software tracing을 activate시키는 역할을 한다. 드라이버는 이 macro 호출이 이루어져야 trace message를 생성할 수 있다.

```
//
// This macro is required to initialize software tracing.
// For Win2K use the deviceobject as the first argument.
//
WPP_INIT_TRACING(pTracedrvDeviceObject,RegistryPath);


//
// This macro is required to initialize software tracing on XP and beyond
// For XP and beyond use the DriverObject as the first argument.
//
WPP_INIT_TRACING(DriverObject,RegistryPath);
```

그리고 windows 2000에서는 WMI를 사용하느냐 하지 않느냐에 따라 달라진다. 자세한 사항은 www.osronline.com에서 다음의 기사를 참고하기 바란다.

WPP Tracing Part1 – Supporting Windows 2000 and Beyond

WPP Tracing Part2 – Coexisting Peacefully with WMILIB

Step 4. Call WPP_CLEANUP from the driver’s Unload routine.

WPP_CLEANUP macro는 드라이버에서 software tracing을 deactivate시킨다. 참고로 이 macro를 호출해주지 않으면 드라이버에 대한 reference가 계속 존재하게 되므로 드라이버가 unload되지 않게 된다. 주의하기 바란다.
Unload routine에서 다음과 같이 호출해 주면 되겠다.

```
//
// Cleanup using DriverObject on XP and beyond.
//
WPP_CLEANUP(DriverObject);

//
// Cleanup using DeviceObject on Win2K. Make sure
// this is same deviceobject that used for initializing.
```

```
//
WPP_CLEANUP(pDevObj);
```

Step 5. Add a RUN_WPP directive to the driver's source file.

WPP는 DDK build환경과 매우 밀접하게 연관되어 있고, RUN_WPP directive는 driver file을 컴파일하기전에 WPP를 호출하는 역할을 한다. 이때 WPP는 WPP macro를 처리하고 필요한 TMH file을 생성하게 된다. 이 TMH file에는 WPP가 생성하는 tracing code에 의해 사용되어지는 함수와 변수들에 대한 선언이 들어있는 txt file이다. 그리고 TMH file에는 드라이버 PDB file에 trace message formatting 명령어를 추가하는 매크로도 포함하고 있다.
참고로 PDB file은 Windows 2000이상의 PE format file들에 대한 debugging information을 가진 file이다. 물론 undocumented file format이다. 참고 교재는 "Undocumented Windows 2000 Secrets"를 보면 도움이 될 것이다.

Tracedrv sample의 sources file을 보면 다음과 같은 내용을 확인할 수 있다.

```
RUN_WPP=  $(SOURCES) -km -gen:{km-w2k.tpl}*.tmh
```

-km : kernel mode를 나타낸다.
-gen : WPP로 하여금 km-w2k.tpl template를 위한 trace message header file을 생성하도록 한다.

Step 6. Add trace message calls to driver code.

Trace message function은 조건이 만족될 때 trace message를 생성시키는 code들이다. 이때 조건은 trace flag와 trace level이 해당된다.
Default WPP trace macro인 **DoTraceMessage**는 trace flag에 기반하여 trace message를 생성한다.

이상의 내용을 우리는 DDK sample "tracedrv"에서 확인할 수 있다.
이제 "tracedrv" sample를 build하면서 위의 내용들과 trace message를 어떻게 확인할 수 있는지를 알아보자.

먼저 build하는 과정을 3단계에 걸쳐 살펴보자.
Step1. Tracedrv를 ddk를 이용하여 build한다.

Step2. tracepdb –f tracedrv.pdb –p Path 로서 해당 path에 TMF file을 만든다. 이때 만들어진 TMF file은 **Tracefmt Tool**이 trace message를 format하는데 사용한다.
Tracepdb tool은 DDK tools₩tracing을 참조하기 바란다.

Step3. 다음과 같이 TMF file의 path를 set한다.
SET TRACE_FORMAT_SEARCH_PATH=Path

위의 과정이 완료되었으면 이제는 trace controller역할을 하는 Tracelog를 구동하여보자 이 tracelog는 log session을 enable, configure, start, update 시키는 역할을 한다. Tracelog역시 DDK tools₩tracing을 참조하기 바란다.

Step1. start software tracing session

```
tracelog –start tracedrv –guid tracedrv.ctl –f tracedrv.etl –flag
0x1
```

위의 명령어는 trace session을 start시키고 이 session을 tracedrv.ctl를 이용하여 우리의 “tracedrv”와 연동시키게 된다. 또한 session buffer로부터 trace message를 받는 tracedrv.etl file을 생성한다. –flag 0x1은 TRACELEVELONE을 set하도록 명시한다.

Step2. test program인 tracectl.exe를 실행시킨다.

이 test program을 실행시킨면 DeviceIoContrl호출을 하게되고 이때 드라이버는 tracing message를 발생시키게 된다. 그리고 이 message는 tracedrv.etl file에 전송되어 진다.

Stop3. 다음의 명령어로서 tracing session을 stop 시킨다.

```
tracelog –stop tracedrv
```

여기까지 진행된 후 이제 드라이버가 생성한 tracing message를 살펴보자. 이때 사용하는 tool은 **Tracefmt**이다. 이 tool역시 DDK tools₩tracing을 참조하기 바란다. 이때 **traceprt.dll**이 필요하다. 이 dll역시 DDK tools₩tracing을 참조하기 바란다.

Step1. trace message를 format한다.

```
tracefmt –o tracedrv.txt –f tracedrv.etl
```

Step2. text editor로 살펴본다.

Tracedrv.txt를 보면 쉽게 이해할 수 있는 다음과 같은 message들이 있음을 볼 수 있다.

[0]0D44.0CF8::03/02/2005-15:18:09.711 [tracedrv]IOCTL = 1

[0]0D44.0CF8::03/02/2005-15:18:09.711 [tracedrv]Hello, 1 Hi

[0]0D44.0CF8::03/02/2005-15:18:09.711 [tracedrv]Hello, 2 Hi

[0]0D44.0CF8::03/02/2005-15:18:09.711 [tracedrv]Hello, 3 Hi

[0]0D44.0CF8::03/02/2005-15:18:09.711 [tracedrv]Hello, 4 Hi

이제는 real time으로 Trace Message를 한 번 살펴보자. 이 기능을 위해서는 TraceView(DDK tools₩tracing₩i386 에 존재)와 mspdb70.dll, msvcr70.dll(이 두 dll은 d아₩bin₩x86에 존재)와 traceprt.dll(이미 언급되었음)이 필요하다.

Step1. Traceview를 실행시킨다.

Step2. Traceview의 File->Create New Log Session

Step3. 우리의 tracedrv를 provider로 등록한다. PDB file 또는 control GUID를 선택한다.

Step4. tracedrv TMF file을 명시한다. (이 부분은 path가 set 되어져 있으면 생략해도된다)

Step5. Real Time Display check box를 선택한 후 finish

Step6. test program tracectl를 실행한다.

맺음말

이상으로 WPP기능을 이용하여 driver 수준에서 tracing message를 발생시키고 그 tracing message를 user mode에서 real-time으로 살펴보는 것까지 알아보았다. 이제 남은 부분은 이 tracing message를 kernel debugger로 redirecting시키는 방법인데 이것은 여러분들의 몫으로 남겨 두겠다. 이 번 기사가 여러분들이 드라이버를 개발함에 있어서 디버깅을 하는데 조금이라도 도움이 되었으면 한다.