

NT 의 Performance Counter 를 이용한 시스템 성능 평가

(1/2) NT 의 성능 평가단, 퍼포먼스 카운터!
(2/2) 보다 진보된 도구, PDH 를 사용하자.

김진홍, aquakim@gmail.com

본 문서는 2000 년 마이크로소프트 월간지에 기고된 것이며, 다른 사람들이 참고할 수 있도록 문서화 한 것입니다. 문서의 내용을 변경하지 않은 상태에서 재 배포 가능합니다.

제 2 회: 보다 진보된 도구, PDH 를 사용하자.

성능 카운터의 저수준 프로그래밍은 까다롭다.
하지만 저수준 프로그래밍에 능숙하지 않은 사람
도 성능 카운터를 얻어낼 수 있다. 바로 PDH 의
도움을 받는 것인데, 심지어는 VB 사용자들도 같
은 PDH 를 이용해 성능카운터를 얻어낼 수 있다.

지난호에서 여러분들은 NT 가 제공하는 퍼포먼스 카운터 데이터를 실제로 분석해보는 시간을 가졌다. NT 가 제공하는 raw 데이터들의 복잡한 구조체를 하나 하나 파헤쳐 들어가며 원하는 값을 얻어내고 또한 그것을 우리가 원하는 형식으로 가공하는 과정을 연습했다. 그런데 그 결과에 비해 과정이 너무 복잡한게 아닌가하는 느낌이 들지 않는가? 정말 배보다 배꼽이 곱절은 컸다고 느끼신 분들이 많을 것이다. 특히 바이너리 구조체에 익숙하지 않은 분들에게는 더욱 복잡하게 느껴지지 않았을까 하는 생각이 든다. 그러나 오르막길이 있으면 뛰어도 힘들지 않은 내리막길이 있는법! 이번호에서는 지난호에서 예고한대로 우리가 연습했던 복잡한 과정들을 대신 수행해 주고 추상화된 고급 인터페이스를 제공해 주는 Performance Data Helper Library(이하 PDH)를 사용할 것이다. 지난호를 이해하신 분들은 이번호의 내용이 식은죽먹기가 아닐까한다. (물론 지난호의 내용을 몰라도 상관은 없다. 그만큼 PDH 는 어렵지 않다는 말이다.) 목표는 NT 시스템의 모든 정보를 얻어내는 것이다. 우리는 지난호에서 CPU 사용율만 -그것도 힘들게- 알아냈었지만, 이번에는 PDH 를 이용하여 기능상 Perfmon 에 버금(?)가는 우리들의 Performance Monitor 를 만들어 보자.

<Counter name string>

PDH 에는 지난호에서 배운 Counter, Instance, Object 말고도 새로운 PDH 용어들이 있다. 먼저 Counter name string 은 우리들이 얻어내고자 하는 Counter 를 지정하는 형식화된 문자열이다.

\\Machine\PerfObject(ParentInstance\ObjectInstance#InstanceIndex)\Counter

기본적으로 위와 같은 형식이며 대소문자를 구분한다. 우리는 그 이름 자체에서도 의미하는 바를 알 수 있는데, Machine 은 Performance Object 들이 존재하는 장치 이름이다. (여기서는 local machine 으로 간주) Local machine 인 경우, '\\Machine' 부분이 필요 없다. PerfObject 는 NT 에서 얻어내려는 Counter 가 존재하는 Object 의 이름이다. 즉, Process 나 System 등을 의미한다. 그 다음에 양 괄호 안에 그 Object 의 Instance 들에 대한 지정을 하게되는데 실제로 ParentInstance 는 접할 일이 거의 없으며, ObjectInstance 만으로 모든 Instance 의 지정이 가능하므로 추가적인 InstanceIndex 는 필요가 없게된다. 다시 정리한 형식은 이와 같다.

\\PerfObject(ObjectInstance)\Counter

Counter 에는 우리가 원하는 Performance Counter 의 이름을 넣게 된다. 만일 System Object 처럼 Instance 를 구분할 필요가 없는 즉, 지난호에 설명한 Single Instance 인 경우, 양 괄호와 ObjectInstance 마저 제거할 수 있다. 모든 Counter 의 이름은 MSDN 을 참조하기 바란다.

한번 Counter name string 을 구성해 보자.

Explorer 프로세스에 얼마나 많은 Handle 이 사용되는지 알아보려면 어떻게 할까. Handle 수에 대한 정보는 Process Object 의 Handle Count 라는 Counter 를 통해 알 수 있다. 그리고 우리가 알고싶은 것은 Explorer 의 Handle 수 이므로, Instance 는 Explorer 가 된다. 그러면 Counter name string 은 이와 같다.

\\Process(Explorer)\Handle Count

우리는 이제 Counter name string 을 구성할 수 있다. 사실 이것이 PDH 에서는 가장 중요한 부분이라고 볼 수 있다.

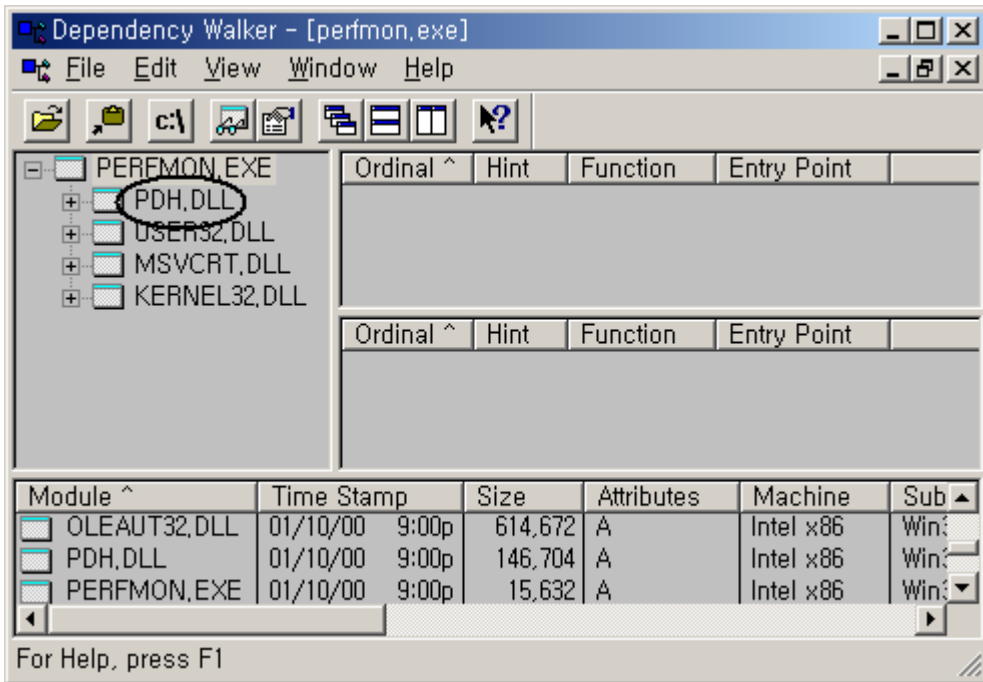
<Query>

간단히 설명하면 Query 는 Counter 들의 집합이다. 우리가 얻어내고자 하는 Counter 는 반드시 Query 에 넣어야 한다. 그리고 나서 PDH 에 Query 를 날려 정보를 얻는다. 일반적인 DataBase 의 Scheme 이라고 볼 수 있다. 우리는 PDH 를 통해 한번에 하나 이상의 서로다른 Counter 들에 대한 정보를 얻을 수 있다. 필요한 모든 Counter 를 Query 에 넣고 PDH 에 날려주면 PDH 는 Query 된 모든 Counter 들의 자료를 얻어내어 준다. 구체적인 과정은 뒤에서 살펴 보자.

<Browsing Performance Counters Dialog Box>

PDH 의 전체적인 작동 과정을 살펴보자. 우선 원하는 Counter 의 Counter name string 을 구성하고 PdhAddCounter()를 이용해 Query 에 넣는다. 그리고 PdhCollectQueryData()를 이용해 Query 로 전달된 모든 Counter 들의 정보를 얻어낸다. 그것을 PdhGetFormattedCounterValue()를 이용해 우리가 쉽게 이할 수 있도록 가공하여 출력하면 된다. 역시 PDH 는 간단하다. 하지만 우리는 PDH 에게 Counter name string 을 제공해야

하며, 그것도 필요한 Counter 의 이름을 알아야만 구성할 수 있다. PDH 는 이 과정 또한 별도의 노력 없이 해결할 수 있도록 도와준다. 그것이 바로 Performance Counters Dialog Box(PDH Dialog Box)이다. 이 Dialog Box 는 NT 에서 기본적으로 제공되는 Perfmon 을 실행시켜 '+' Icon 을 눌렀을 때 나타나는 Dialog Box 와 같다. 그것은 Perfmon 역시 PDH 를 사용했기 때문이다. 이 Dialog Box 는 가능한 모든 Object, Counter, Instance 들을 나열해 주며, 사용자가 직접 원하는 Counter 를 선택할 수 있도록 해준다. 선택시 PDH 는 자동적으로 Counter name string 을 구성해 준다. 이 Dialog Box 는 PdhBrowseCounters()라는 PDH API 를 통해 띄운다. 그러나 이 API 는 Dialog Box 를 띄우기 위해 필요한 세부 정보가 들어있는 PDH_BROWSE_DLG_CONFIG 구조체가 필요하다. <표 1> 참조.



<화면 1> Perfmon 이 PDH.DLL 을 사용하는 모습

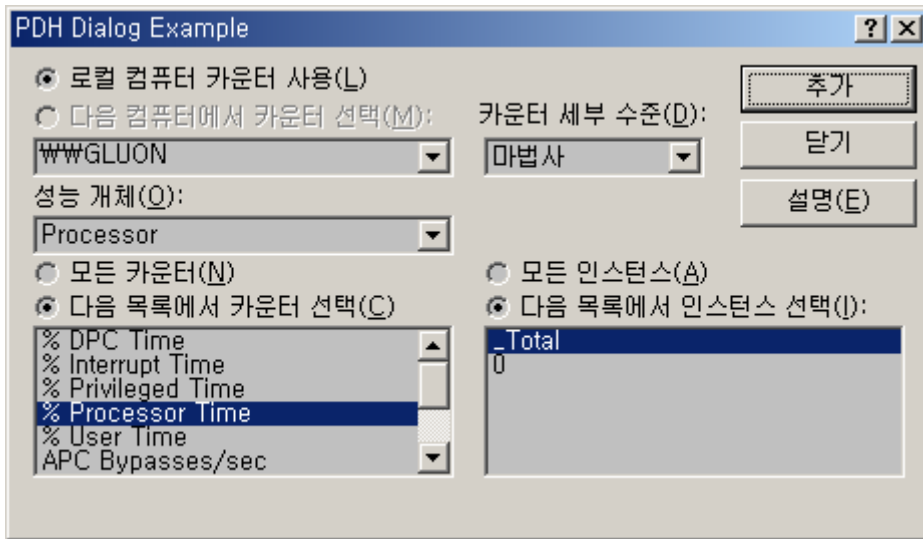
<표 1> PDH_BROWSE_DLG_CONFIG 구조체

멤버변수	설명
bIncludeInstanceIndex	Counter name string 에 InstanceIndex 필드 포함 결정. TRUE: 포함
bSingleCounterPerAdd	PDH Dialog Box 의 [추가] 단추를 누를 때 동시에 여러 개의 Counter 들을 선택할 수 있는가 결정. TRUE: 한번에 하나, FALSE: 한번에 여러 개 가능.
bSingleCounterPerDialog	PDH Dialog Box 의 [추가](확인) 단추를 누를 때 Dialog Box 가 Close 될 것인지 결정. TRUE: [추가](확인) 단추를 누르면 Dialog Box 를 Close 함. FALSE: [닫기] 버튼을 누를때까지 Close 되지 않음.
bLocalCountersOnly	원격 machine 의 Counter 에 접근 가능한지 결정. TRUE: Local Counters Only.

bWildcardInstances	Counter name string 의 Instance 필드 대신 WildCard 를 사용할 수 있는지 결정. TRUE: All Instances 를 선택시 Instance 필드에 '*'가 들어감. FALSE: 선택된 모든 Instance 가 MULTI_SZ 문자열 형식으로 Instance 필드에 채워짐.
bHideDetailBox	PDH Dialog Box 에서 “Detail Level”을 제거할 것인지 결정. TRUE: “Detail Level” Combo Box 제거.
blInitializePath	PDH Dialog Box 가 표시될 때 기본적으로 어떤 Counter 가 선택되는지를 결정. TRUE: szReturnPathBuffer 로 전달되는 첫 번째 Counter 와 Object 를 표시. FALSE: machine 이 돌려주는 기본 Counter 와 Object 를 표시.
bReserved	사용안함. 0 으로 초기화.
hWndOwner	PDH Dialog Box 의 부모 윈도우를 설정. 없으면 NULL.
szDataSource	사용안함. 반드시 NULL 로 초기화.
szReturnPathBuffer	Counter name string 이 들어갈 버퍼 설정.
cchReturnPathLength	szReturnPathBuffer 에 설정한 버퍼의 최대 크기.
pCallback	PDH Dialog 가 호출하는 callback 함수 설정.
dwCallbackArg	호출되는 callback 함수에 전달되는 값. 임의로 사용 가능하다.
CallbackStatus	PDH Dialog 의 상태가 전달되는 멤버 변수.
dwDefaultDetailLevel	PDH Dialog Box 의 “Detail Level”을 설정. bHideDetailBox 가 TRUE 이면, 이 멤버 변수는 표시될 Counter 들을 필터링함. 그렇지 않으면 이 멤버 변수는 PDH Dialog Box 의 초기 “Detail Level” 로 사용됨.
szDialogBoxCaption	PDH Dialog Box 의 Caption 으로 사용될 문자열을 가리키는 멤버 변수.

<표 1>을 보자. 필드가 상당히 많은 편이다. 그런데 먼저 잡고 넘어갈 것이 있다. 바로 MSDN 의 내용과 실제 필드의 내용에 차이가 있다는 점이다. 우선 MSDN 에서는 **bSingleCounterPerAdd** 의 TRUE 와 FALSE 의 의미가 뒤바뀌어 있다. 그리고 **szDataSource** 이라는 멤버 변수는 MSDN 에는 나와 있지 않고 대신 **szReserved** 만 있을 뿐이다. 그러나 실제 구조체 안에는 **szReserved** 라는 멤버변수는 없다. 나중에 이름이 **szDataSource** 로 바뀐 것 같은데, 어떤 목적인지는 모르지만 역시 사용되지는 않는다. MS 사의 실수가 아닌가 싶다. 필자는 VC 6.0 을 사용하는데 5.0 이하는 MSDN 의 내용과 동일 한지는 확인하지 못했다.

위와 같이 필드를 채우고 나서 PdhBrowseCounters() API 를 부르면 PDH Dialog Box 가 뜨 게 된다.



<화면 2> PDH Dialog Box

<리스트 1> PDH Dialog Box 를 띄우는 기본 코드

// 주의: 아래 코드는 Win32 Console Application 으로 제작해야 함.

```
#include <windows.h>
#include <pdh.h>
#include <tchar.h>
#include <stdio.h>

#define INIT_RETURNPATH_SIZE 1024 // 초기 버퍼 크기
DWORD   gdwReturnPathSize;
TCHAR   *gszReturnPath;
PDH_BROWSE_DLG_CONFIG   pdhconf;

// PDH Dialog Box 가 호출하는 callback 함수
PDH_STATUS __stdcall pdh_callback(DWORD)
// 팁: 변수 이름을 제거하면 사용되지 않는 변수라는 warning 메시지가 사라진다.
{
    return ERROR_SUCCESS;
}

int initialize()
{
    // Counter name string 을 담을 버퍼 설정
    gdwReturnPathSize = INIT_RETURNPATH_SIZE;
    gszReturnPath = (TCHAR*)VirtualAlloc(NULL, gdwReturnPathSize * sizeof(TCHAR),
                                         MEM_COMMIT, PAGE_READWRITE);

    if (!gszReturnPath)
    {
        MessageBox(NULL, "Not enough memory", "System", MB_OK);
        return FALSE;
    }

    return TRUE;
}

// #define PERF_DETAIL_NOVICE      100 // The uninformed can understand it
// #define PERF_DETAIL_ADVANCED   200 // For the advanced user
// #define PERF_DETAIL_EXPERT     300 // For the expert user
// #define PERF_DETAIL_WIZARD     400 // For the system designer
```

```

void pdh_init(HWND hwnd, TCHAR *buff, DWORD buffsize)
{
    ZeroMemory((void*)gszReturnPath, gdwReturnPathSize * sizeof(TCHAR));
    // PDH_BROWSE_DLG_CONFIG 설정
    pdhconf.bIncludeInstanceIndex = FALSE;
    pdhconf.bSingleCounterPerAdd = TRUE;
    pdhconf.bSingleCounterPerDialog = TRUE;
    pdhconf.bLocalCountersOnly = TRUE;
    pdhconf.bWildCardInstances = FALSE;
    pdhconf.bHideDetailBox = FALSE;
    pdhconf.bInitializePath = FALSE;
    pdhconf.bReserved = NULL;
    pdhconf.hWndOwner = hwnd;
    pdhconf.szDataSource = NULL;
    pdhconf.szReturnPathBuffer = (LPTSTR)buff;
    pdhconf.cchReturnPathLength = buffsize;
    pdhconf.pCallBack = pdh_callback;
    pdhconf.dwCallBackArg = 0;
    pdhconf.dwDefaultDetailLevel = PERF_DETAIL_WIZARD;
    pdhconf.szDialogBoxCaption = (LPTSTR)"PDH Dialog Example";
}

void uninitialize()
{
    VirtualFree(gszReturnPath, 0, MEM_RELEASE);
}

void main()
{
    if (initialize() == FALSE) exit(1);

    // PDH_BROWSE_DLG_CONFIG 구조체 초기화
    pdh_init(NULL, gszReturnPath, gdwReturnPathSize);
    // PDH Dialog Box 띄움
    PdhBrowseCounters(&pdhconf);
    // PDH 가 만들어준 Counter name string 출력
    _tprintf("The counter string is :%Wn  %sWn", gszReturnPath);
    uninitialize();
}

```

<리스트 1>을 보자. Dialog Box 를 띄우는데 꽤 긴 코드가 들어간다. 하지만 나머지를 완성하는데 여기서 조금씩만 추가하면 되므로 크게 길어지지는 않는다. initialize()에서는 Counter name string 에 필요한 초기 버퍼를 준비한다. 그리고 PdhBrowseCounters() API 를 사용하기 전에 필요한 구조체를 pdh_init()에서 초기화 한다. <리스트 1>은 Console Application 이므로 PDH Dialog Box 의 hWndOwner 에 NULL 을 넣었다. PDH Dialog Box 를 띄운 후 원하는 Counter 를 추가하게 되면 PDH 는 지정된 callback 함수를 불러 필요한 기능을 처리하도록 한다. callback 함수의 타입은 이와 같다.

```

PDH_STATUS __stdcall CounterPathCallback(
    IN DWORD dwArg // Pointer to a PDH_BROWSE_DLG_CONFIG structure
);

```

<리스트 1>에서 pdh_callback()이 PDH 가 호출하는 callback 함수이다. 아직은 이 함수에 아무 기능도 넣지 않았다. 이제 우리가 원하는 기능을 추가해 보자. 위에서 언급했듯이 QUERY 에 원하는 Counter 를 PdhAddCounter() API 를 통해 추가해야 하는데, PdhAddCounter()는 Counter 의 Full Path(Counter name string)를 요구한다. PDH Dialog 는 사용자가 Counter 를 선택하면 그 Counter 의 Full Path 를 szReturnPathBuffer 가 가리키는 버퍼에 담아주고 callback 함수를 호출한다. 이제 callback 함수에서 Counter name string

즉, Counter 의 Full Path 를 PdhAddCounter()에 넘겨주면 된다.
 그런데 여기서 고려해야 할 것이 있다. 지난호에서 언급했듯이 NT 는 필요한 버퍼의 크기를 미리 말해주지 않는다. 그래서 PDH 도 그것을 알 수가 없는데, 우리가 szReturnPathBuffer 를 통해 넘겨준 버퍼가 충분하지 못하면 PDH 는 callback 함수를 호출할 때 자신의 상태를 나타내는 CallBackStatus 멤버변수에 버퍼가 모자라다는 뜻의 PDH_MORE_DATA 메시지를 전달해준다. PDH 의 메시지들은 pdhmsg.h 에 있다.

<리스트 2> 새로운 callback 함수

```
PDH_STATUS __stdcall pdh_callback(DWORD)
{
    if (pdhconf.CallBackStatus == PDH_MORE_DATA)
    {
        // 기존 버퍼 제거
        if (VirtualFree(gszReturnPath, 0, MEM_RELEASE) == NULL)
            return PDH_MEMORY_ALLOCATION_FAILURE;
        // 버퍼의 크기를 일정한 만큼 늘림
        gdwReturnPathSize += BUFFER_INCREMENT_SIZE;
        gszReturnPath = (TCHAR*)VirtualAlloc(NULL, gdwReturnPathSize * sizeof(TCHAR),
            MEM_COMMIT, PAGE_READWRITE);

        if (gszReturnPath == NULL)
            return PDH_MEMORY_ALLOCATION_FAILURE;

        pdhconf.szReturnPathBuffer = gszReturnPath;
        pdhconf.cchReturnPathLength = gdwReturnPathSize;

        return PDH_RETRY;
    }
}
```

```
TCHAR *curPath;
int len;

curPath = gszReturnPath;
while (1)
{
    len = lstrlen(curPath);
    if (len == 0) break;
    // PDH 가 만들어준 Counter name string 을 출력
    _tprintf("%s\n", curPath);
    // 카운터를 동시에 여러 개 선택할 경우 MULTI_SZ 로 구성되므로 반복한다.
    curPath += (len + 1);
}
}
```

```
return ERROR_SUCCESS;
}
```

<리스트 2>에서 처럼 PDH 가 PDH_MORE_DATA 라는 메시지를 전달하면 적당히 버퍼를 늘려야 하며, PDH_BROWSE_DLG_CONFIG 의 szReturnPathBuffer 와 cchReturnPathLength 를 새로운 내용으로 다시 설정해 주고 PDH_RETRY 를 리턴해야 한다. <리스트 2>의 박스 부분을 보면 PDH 가 만들어준 Counter name string 을 (MULTI_SZ 인것을 고려해) 화면에 출력해 주는 부분을 볼 수 있다. 나중에 이 부분을 PdhAddCounter() API 로 대체하게 된다. 그리고 Counter 의 다중 선택도 가능하므로 PDH_BROWSE_DLG_CONFIG 의 bSingleCounterPerAdd 와 bSingleCounterPerDialog 를 둘 다 FALSE 로 설정한다. PDH 에는 선택한 Counter 에 대한 설명이 담긴 문자열을 돌려주는 기능도 있다. 이것은 PdhGetCounterInfo() API 를 통해 얻을 수 있다. 이것은 사실 PDH 가 만들어 주는 것이 아니라 NT 가 만들어 주는 것이다. 지난호에서도 Counter 에 대한 정보를 얻을 수 있었지만, 안그래도 복잡한데 더 복잡해질까봐 넣지 않았었다. 이제 실제로 PdhAddCounter()를 설치해보자.

<리스트 3> PdhAddCounter()가 설치된 callback 함수

```
#define BUFFER_INCREMENT_SIZE 100
#define INIT_RETURNPATH_SIZE 100

HQUERY hQuery;
typedef struct
{
    TCHAR pCounterName[100]; // 카운터의 이름을 저장할 변수
    HCOUNTER hCounter; // 실제 카운터 데이터를 담을 변수
} COUNTERSTRUCT;

// 동시에 100 개 이상 관찰할 일이 있을까?
#define COUNTER_MAX 100

COUNTERSTRUCT cs[COUNTER_MAX];
int cs_idx = 0; // COUNTERSTRUCT index

PDH_STATUS __stdcall pdh_callback(DWORD)
{
    ... (생략) ...
    TCHAR *curPath;
    int len;
    PDH_COUNTER_INFO *ppci;
    DWORD pci_len;

    pci_len = 2048;
    ppci = (PDH_COUNTER_INFO*)VirtualAlloc(NULL, pci_len,
        MEM_COMMIT, PAGE_READWRITE);

    curPath = gszReturnPath;
    while (1)
    {
        pci_len = 2048;

        len = lstrlen(curPath);
        if (len == 0) break;

        // 카운터 이름 출력
        _tprintf("%sWn", curPath);

        // 더 이상 카운터를 추가할 수 없으면 skip.
        if (cs_idx == COUNTER_MAX)
        {
            _tprintf("No more counter availableWn");
            break;
        }

        // COUNTERSTRUCT 에 카운터 이름을 저장
        lstrcpy(cs[cs_idx].pCounterName, curPath);

        // add counter
        if (PdhAddCounter(hQuery, (LPCTSTR)curPath,
            0, &(cs[cs_idx].hCounter)) != ERROR_SUCCESS)
            exit(1);

        // 카운터 정보 출력
        PdhGetCounterInfo(cs[cs_idx].hCounter, TRUE, &pci_len, ppci);
        _tprintf("Explain: %sWn", ppci->szExplainText);
    }
}
```

```

        cs_idx++;
        curPath += (len + 1);
    }

    // free PDH_COUNTER_INFO buffer
    VirtualFree(ppci, 0, MEM_RELEASE);

    return ERROR_SUCCESS;
}

```

<리스트 3>의 callback 함수에는 우리가 원하는 기능을 모두 넣었다. PDH Dialog Box 에서 사용자가 원하는 Counter 를 선택하면 PDH 는 이 callback 함수를 호출하고 이 함수에서는 PDH 가 만들어준 Counter name string 을 분석해 QUERY 에 추가한다. 동시에 추가된 Counter 에 대한 정보도 출력하게 해 놓았다. PdhGetCounterInfo() API 에 대한 자세한 설명은 MSDN 을 참고하기 바란다.

PdhAddCounter()를 이용해 QUERY 에 Counter 를 추가하려면 미리 QUERY 를 Open 해야 한다.

```
if (PdhOpenQuery(NULL, 0, &hQuery) != ERROR_SUCCESS) exit(1);
```

위와 같이 Query 를 Open 한다. 이제 PdhOpenQuery() 를 사용할 수 있다.

<Updating Performance Counter Data>

Query 에 있는 Counter 들의 실제 성능 데이터를 얻기위해 PdhCollectQueryData() API 를 호출한다. 이 함수를 호출하면 Query 에 Add 할때 지정된 HCOUNTER 타입의 Counter 들 변수에 성능 데이터가 들어가게 된다. 그러나 성능 데이터는 raw 데이터라 우리가 이해할 수 있는 형식으로 변환해야 한다. 그것을 PdhGetFormattedCounterValue() API 가 해준다. <리스트 4>를 보자.

<리스트 4> Performance Data 를 Update 하는 부분

```

int pdh_update(long *ret, int idx)
{
    PDH_FMT_COUNTERVALUE pdhFmtvalue;

    if (PdhCollectQueryData(hQuery) == ERROR_SUCCESS)
        if (PdhGetFormattedCounterValue(cs[idx].hCounter,
            PDH_FMT_LONG,
            NULL,
            &pdhFmtvalue) == ERROR_SUCCESS)
        {
            *ret = pdhFmtvalue.longValue;
            return TRUE;
        }

    return FALSE;
}

```

<리스트 4>를 통해 Query 에 Add 된 Counter 들의 성능 데이터를 얻어오고 우리가 이해하는 타입으로 바꾼다. 일반적으로 LONG 타입으로 변환하면 되는데, PdhGetFormattedCounterValue()의 첫번째 인자로 들어가는 PDH_FMT_LONG 이 그것이다.

raw 데이터는 변환되어 PDH_FMT_COUNTERVALUE 구조체 변수로 들어가는데 그 구조는 <표 2>와 같다.

<표 2> PDH_FMT_COUNTERVALUE 구조체

멤버변수	설명
CStatus	Counter의 상태 변수
union { LONG longValue; double doubleValue; LONGLONG largeValue; };	타입 변환된 Counter 값을 저장하는 변수. union으로 되어 있어 원하는 형식으로 읽을 수 있다. 여기서는 모두 longValue를 통해 LONG형식의 값을 읽는다.

<리스트 4>의 pdh_update() 함수를 통해 원하는 값을 얻고 그것을 출력하면 전체적인 구조는 완성된 것이다.

```

"C:\project\PDHExam\Debug\PDHExam.exe"
#Processor<_Total>% Processor Time
Explain: % Processor Time은 프로세서가 비유휴 스레드를 실행하는 시간의 백분율입니다. 이 카운터는 프로세서 활동의 주요 표시기로 만들어졌습니다. 이것은 프로세서가 각 샘플 간격 동안 유휴 프로세스의 스레드를 실행하는 데 소비한 시간을 측정하여 100%에서 그 값을 뺀 것입니다. <각 프로세서에는 유휴 스레드가 있는데 이것은 다른 어떤 스레드도 실행되지 않을 때 사이클을 소
#Processor<_Total>% Processor Time : 99

#Processor<_Total>% Processor Time : 8

#Processor<_Total>% Processor Time : 0

#Processor<_Total>% Processor Time : 0

#Processor<_Total>% Processor Time : 0

-
  
```

<화면 3> PDH Dialog Box 를 닫은 후 실행되는 화면

<리스트 5> 완성된 코드의 main()

```

void main()
{
    int count = 0;
  
```

```

LONG val;

if (initialize() == FALSE) exit(1); // 초기화

// Query open
if (PdhOpenQuery(NULL, 0, &hQuery) != ERROR_SUCCESS) exit(1);

// PDH_BROWSE_DLG_CONFIG 설정
pdh_init(NULL, gszReturnPath, gdwReturnPathSize);
// PDH Dialog Box 생성
PdhBrowseCounters(&pdhconf);

while (count++ < 20)
{
    for (int i=0; i<cs_idx; i++)
    {
        // Counter 이름 출력
        _tprintf("%s : ", cs[i].pCounterName);
        // Counter 값을 얻어내는데 에러가 있으면 while loop 을 탈출
        if (pdh_update(&val, i) == FALSE)
        {
            count = 100;
            continue;
        }
        // 실제 Performance Counter 값 출력
        _tprintf("%dWn", val);
    }

    _tprintf("Wn");
    Sleep(500);
}

// Query close
PdhCloseQuery(hQuery);
// 할당된 메모리 해제
uninitialize();
}

```

<리스트 5>에서 Performance Counter 값을 출력한 후에 0.5 초간 Sleep() 하도록 해 놓았다. 0.5 초는 태스크 매니저의 업데이트 속도에서 빠름에 해당한다. 이것보다 더 빠르게 측정해볼 수도 있겠지만 너무 빠른 주기로 측정하면 (특히 CPU Load 등에서) PDH 자체가 시스템에 주는 부하로 인해 정확한 측정이 어렵게 된다. 그래서 일반적으로 느린 주기의 측정을 하게 되는데, 이때 주의 할 것은 update 된 순간의 Counter 수치를 그대로 믿고 사용하지 말고 반드시 여러 번 측정하여 평균한 데이터를 사용해야 한다는 것이다. 호기심 많은 독자중에서는 같은 Counter 를 두 번 이상 [추가] 하여 측정해 볼 수도 있겠는데, 바로 알게되지만 제대로 된 값을 얻지 못하게 된다. 동시에 여러 개의 Counter 를 측정할 때 그 Counter 들간에는 같은 것이 없어야 정확한 결과를 얻을 수 있다.

언급한대로, 정확한 측정을 위해서는 반복 측정한 결과를 평균해야 한다. PDH 에서는 통계에 관련된 함수또한 마련해 주고 있다. PdhComputeCounterStatistics() API 가 그것이다. 이름 길이 만큼이나 파라메타들이 복잡해 보인다. 그러나 기능은 매우 간단하다. 사용자가 PDH 규칙에 따라 별도로 제공하는 Queue 형식의 Counter 측정 기록들을 넣어주면 그 기록들 중 최대, 최소값과 평균값을 제공해줄 뿐이다. 필자가 생각하기에는 이 함수를 사용하는 것보다 여러분들이 그냥 평균내서 사용하는 것이 더 편할 것 같다.

자. PDH 로 여러분은 많은 것을 했다. 그리고 PDH 의 중요한 부분은 모두 배운셈이다.

Visual Basic 사용자들도 PDH를 사용할 수 있다. PDH.DLL에는 VB용 함수들도 정의되어 있다. 예를들면 PdhAddCounter는 PdhVbAddCounter와 같이 Pdh다음에 Vb가 붙은 형태로 존재한다. (PdhOpenQuery, PdhCloseQuery와 같이 몇몇 API는 특별히 Vb용으로 따로 존재하지는 않지만 Vb에서도 사용할 수 있다.) VB에서 PDH를 이용한 예제는 1998년 MSJ 3월호의 UnderTheHood란에서 받아 볼 수 있다. URL은 <http://www.microsoft.com/msj/0398/hood0398.htm> 이다.

<Enumeration>

마지막으로 중요한 기능 하나를 배워보자. 바로 Enumeration 기능이다. 지난호에서처럼 우리가 직접 저수준 코딩을 하게되면 NT에게 모든 Counter 정보를 요청해서 navigation 할 수 있겠지만, PDH는 저수준 정보를 모두 감추고 있으므로 PDH에서 제공하는 Enumeration API를 사용해야 한다.

Enumeration을 적용해볼 가장 적합한 예를 들어보자. 현재 내 시스템에서 활동중인 모든 프로세스의 이름과 그 Process ID를 얻고 싶다면 어떻게 할까. (필자처럼 이것이 궁금했던 독자들도 있을것이다.) 다른 방법으로 Process의 목록이나 ID등을 얻을 수도 있겠지만, PDH가 제공하는 Enumeration 기능을 활용하면 쉽게 그 결과를 얻을 수 있다. 그 결과를 얻기위해 PDH Dialog Box를 띄워 Process Object를 선택해서 그 안의 모든 Instance를 선택해 볼수도 있겠다. 하지만 매번 수작업으로 Dialog Box를 조작할 수도 없는 노릇이다. 결국 이 경우 PDH Dialog Box는 쓸모가 없게되고, PDH 대신 모든 프로세스를 얻고(Enumerating) 자동으로 Counter name string을 구성해내야 한다. <리스트 6>에 그 해답이 있다.

<리스트 6> PDH Enumeration

```
void EnumAllProcessID()
{
    PDH_STATUS   pdhStatus           = NULL;
    LPTSTR       szCounterListBuffer = NULL;
    DWORD        dwCounterListSize   = 0;
    LPTSTR       szInstanceListBuffer = NULL;
    DWORD        dwInstanceListSize  = 0;
    LPTSTR       pCurInstance       = NULL;
    PDH_COUNTER_PATH_ELEMENTS   cpe;
    TCHAR        szFullPath[100];
    DWORD        dwFullPathSize = 100 * sizeof(TCHAR);

    // 필요한 버퍼 결정
    (1) pdhStatus = PdhEnumObjectItems(
        NULL, // reserved
        NULL, // local machine
        TEXT("Process"), // enumerate 할 object
        NULL, // 버퍼의 크기를 알고자할때는 필요없다.
        &dwCounterListSize, // 버퍼의 크기를 알려달라는 뜻
        NULL,
        &dwInstanceListSize,
        PERF_DETAIL_WIZARD, // counter detail level
        0);

    if (pdhStatus != ERROR_SUCCESS)
        return;
}
```

```

// PdhEnumObjectItems()가 알려준 버퍼 크기만큼 메모리를 할당함.
szCounterListBuffer = (TCHAR*)VirtualAlloc(NULL, dwCounterListSize * sizeof(TCHAR),
MEM_COMMIT, PAGE_READWRITE);

szInstanceListBuffer = (TCHAR*)VirtualAlloc(NULL, dwInstanceListSize * sizeof(TCHAR),
MEM_COMMIT, PAGE_READWRITE);

(2) pdhStatus = PdhEnumObjectItems(
    NULL, // reserved
    NULL, // local machine
    TEXT("Process"), // enumerate 할 object
    szCounterListBuffer,
    &dwCounterListSize,
    szInstanceListBuffer,
    &dwInstanceListSize,
    PERF_DETAIL_WIZARD, // counter detail level
    0);

if (pdhStatus != ERROR_SUCCESS)
    return;

for (pCurInstance = szInstanceListBuffer;
    *pCurInstance != 0;
    pCurInstance += strlen(pCurInstance) + 1)
{
    dwFullPathSize = 100 * sizeof(TCHAR);

(3) // full counter path 생성
    cpe.szMachineName = NULL;
    cpe.szObjectName = TEXT("Process");
    cpe.szInstanceName = pCurInstance;
    cpe.szParentInstance = NULL;
    cpe.dwInstanceIndex = -1;
    cpe.szCounterName = TEXT("ID Process");
    PdhMakeCounterPath(&cpe, szFullPath, &dwFullPathSize, 0);

    // 더 이상 카운터를 추가할 수 없으면 skip.
    if (cs_idx == COUNTER_MAX)
    {
        _tprintf("No more counter available\n");
        break;
    }

    // COUNTERSTRUCT 에 카운터 이름을 저장
    lstrcpy(cs[cs_idx].pCounterName, szFullPath);

(4) // add counter
    if (PdhAddCounter(hQuery, (LPCTSTR)szFullPath,
        0, &(cs[cs_idx].hCounter)) != ERROR_SUCCESS)
        break;

    cs_idx++;
}

VirtualFree(szCounterListBuffer, 0, MEM_RELEASE);
VirtualFree(szInstanceListBuffer, 0, MEM_RELEASE);
}

```

Enumeration 을 하는 <리스트 6>를 보면 PDH Dialog Box 와 callback 함수가 해야할 일들 을 모두 하고 있다. 덕분에 callback 함수와 Dialog Box 를 초기화 하는 부분이 필요없게

된다. 우리는 시스템에 상주하는 모든 Process 의 이름과 ID 를 얻고자 한다. Process Object 의 Instance 이름이 바로 Process 의 이름이다. 결국 Process Object 를 Enumeration 해서 모든 Instance 의 목록을 얻어야 한다. 그런데 문제가 있다. 여기서도 Enumeration 시 필요한 버퍼의 크기가 어떻게 되는지 미리 알 수가 없다. (1) 부분의 PdhEnumObjectItems()를 보자. 3 번째 인자에 Enumeration 을 원하는 Object 의 이름을 넣었다. 5 번째와 7 번째 인자로 전달한 변수의 값은 0 이다. 0 으로 초기화된 변수를 5 번째 7 번째 전달하면 그곳에 필요한 버퍼의 최소 크기가 얼마인지를 돌려준다. 결국 한번은 이렇게 해서 버퍼의 크기를 알아내야 하고 (2) 에서 처럼 다시 Enumeration 을 해야한다.

Process 의 이름을 얻어내었다. 프로세스의 ID 를 얻자. 프로세스의 ID 에 해당하는 Counter 는 'ID Process' 이다. 우리는 Instance 의 이름과 Counter 의 이름을 알고 있으므로 이제 Counter name string 의 Full Path 를 구성할 수 있다. PDH 에는 Full Path 를 자동으로 구성해 주는 함수가 존재한다. PdhMakeCounterPath() API 가 그것이다. 이 API 를 사용하기 위해서는 PDH_COUNTER_PATH_ELEMENTS 라는 구조체를 준비해야 한다. 정말로 구조체가 난무하고 있다. 하지만 이름만 길지 어렵지 않으니 조금만 더 인내심을 갖기 바란다. <표 3>에 그 구조체의 설명이 있다.

<표 3> PDH_COUNTER_PATH_ELEMENTS 구조체

멤버변수	설명
szMachineName	Performance Counter Object 가 존재하는 machine 의 이름. 여기서는 Local machine 으로 간주하므로 NULL 로 하면 되겠다. (optional)
szObjectName	이름 그대로 Object 의 이름. (required)
szInstanceName	Instance 의 이름. (optional)
szParentInstance	부모 Instance 의 이름. (optional)
dwInstanceIndex	Index 번호. (optional) Index 번호가 없을때에는 -1 로 설정
szCounterName	Counter 의 이름. (required)

(3) 부분에서 <표 3>의 구조체 필드를 완성하고 PdhMakeCounterPath() API 를 통해 Counter name string 의 Full Path 를 완성한다. (반대로, Full Path 에서 원하는 부분을 취하는 함수인 PdhParseCounterPath(), PdhParseInstanceName() API 도 있다.) 이제 마지막으로 (4) 에서 PdhAddCounter() API 를 수행하면 필요한 것을 다 한것이다. 이제 화면에 표시만 하면 되는데, Process 의 이름과 ID 는 한번만 출력하면 되므로, <리스트 5>에서 pdh_init(), PdhBrowseCounters(), while loop 을 제거하면된다.

```
"C:\project\PDHEXAM\Debug\PDHEXAM.exe"
#Process<Idle>#ID Process : 0
#Process<System>#ID Process : 8
#Process<smss>#ID Process : 136
#Process<csrss>#ID Process : 164
#Process<winlogon>#ID Process : 184
#Process<services>#ID Process : 212
#Process<lsass>#ID Process : 224
#Process<svchost>#ID Process : 388
#Process<SPoolSU>#ID Process : 416
#Process<svchost>#ID Process : 388
#Process<regsvc>#ID Process : 484
#Process<mstask>#ID Process : 500
#Process<winmgmt>#ID Process : 548
#Process<mspmbspv>#ID Process : 572
#Process<Explorer>#ID Process : 784
#Process<point32>#ID Process : 624
#Process<msdev>#ID Process : 844
#Process<conime>#ID Process : 980
#Process<WINWORD>#ID Process : 1040
#Process<winamp>#ID Process : 1128
#Process<vcspawn>#ID Process : 1264
#Process<PDHEXAM>#ID Process : 1160
#Process<_Total>#ID Process : 0
Press any key to continue.
```

<화면 4> 필자 시스템의 Process 들을 Enumerate 한 화면

자. 수고를 하셨다. 여러분은 지금까지 Counter name string 이 무엇인지, 또 그것을 PDH Dialog 를 이용해 어떻게 자동으로 구성하는지, 어떻게 Performance Counter 값을 PDH 로부터 얻어내는지, 그것을 또 어떻게 가공하는지에 대한 방법들을 배웠다. 그리고 마지막으로 Enumeration 이란 무엇이고 어떻게 사용하는지에 대하여 배웠다. 이제 여러분은 Perfmon 과 태스크매니저가 가지고 있는 신기한(?) 능력들을 모두 습득한 것이다. 비록 여기서 만들어본 예제가 Console Application 이라 인터페이스는 보잘 것 없으나, 그것을 어떻게 꾸미는지 그리고 어떻게 응용하는지는 이제 여러분의 몫이라 하겠다.

그런데 만능으로 보이는 PDH 도 못하는 것이 하나 있다. 그것은 Custom Performance Counter 를 시스템에 설치하는 방법이다. PDH 는 못하지만 지난호에 배운 저수준의 접근 방법으로는 가능하다. 하지만 실제로 사용하게 될 일이 별로 없기 때문에 꼭 배워야 할 필요는 없다. 참고로 약간의 COM 의 도움을 받아 Custom Performance Counter 를 설치할 수도 있다. 그것은 참고문헌을 찾아보기 바란다.

이번호까지 2 회의 분량으로 NT 나 Windows 2000 에서 제공하는 Performance Counter 를 살펴보았다. 비록 모든 부분을 파헤쳐 보지는 않았지만 그래도 필요한 것은 다 했다고 생각한다. 날씨도 더운데 많이들 수고하셨다. 그럼 마소를 통해 다음에 또 다시 독자들을 만나게 될것을 기약하며 이만 줄이겠다.

참고문헌

[1] “ Design Your Application to Manage Performance Data Logs Using the PDH Library” ,
MSJ, December 1999,
<http://www.microsoft.com/msj/defaulttop.asp?page=/msj/1299/pdh/pdhtop.htm>

[2] “ Instrumenting Windows NT Applications with Performance Monitor” , MSDN

[3] “ It’ s Simple to Build PerfMon Support into Your Apps With a Little Help from COM” , MSJ, February 2000, <http://www.microsoft.com/msj/0200/comperf/comperf.asp>