치트 엔진과 OllyDbg는 게임의 메모리(Memory)와 코드(Code)를 분리하는 데 도움이될 수 있지만 게임이 파일(Files), 레지스트리 값(Registry values), 네트워크 연결(Network connections) 및 기타 프로세스(other processes)와 상호 작용하는 방식도 이해해야 합니다. 이러한 상호 작용이 작동하는 방식을 배우려면 프로세스의 외부 작업(external actions)을 모니터링 하는 데 탁월한 두 가지 도구인 프로세스 모니터(Process Monitor)와 프로세스 탐색기(Process Explorer)를 사용해야 합니다. 이러한 도구를 사용하여 전체 게임 맵(game map)을 추적하고, 저장 파일(save files)을 찾고, 설정을 저장하는 데 사용되는 레지스트리 키를 식별하고, 원격 게임 서버의 인터넷 프로토콜(IP) 주소를 열거할 수 있습니다.

이 장에서는 Process Monitor와 Process Explorer를 모두 사용하여 시스템 이벤트를 기록하고 게임이 어떻게 관련되었는지 확인하기 위해 검사하는 방법을 알려드리겠습니다. 주로 초기 정찰에 유용한 이러한 도구는 게임이 시스템(OS)와 상호 작용하는 방식에 대한 명확하고 장황한 그림을 제공합니다.

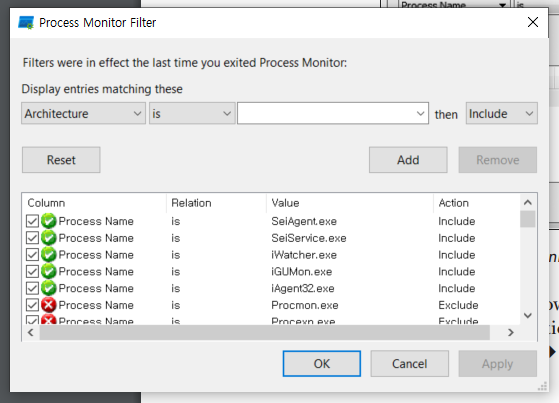
**Process Monitor**

게임이 레지스트리(Registry), 파일 시스템(File system) 및 네트워크(Network)와 상호 작용하는 방식을 살펴보는 것만으로도 게임에 대해 많은 것을 배울 수 있습니다. Process Monitor는 이러한 이벤트를 실시간으로 기록하고 데이터를 디버깅 세션에 원활하게 통합할 수 있는 강력한 시스템 모니터링 도구입니다. 이 도구는 게임과 외부 환경의 상호 작용에 관한 방대한 양의 유용한 데이터를 제공합니다. 계산된 검토(때로는 자발적인 직관)를 통해 이 데이터는 게임 작동 방식을 보고 조작하는 데 도움이 되는 데이터 파일(Data files), 네트워크 연결(Network connections) 및 레지스트리(Registry) 이벤트에 대한 세부 정보를 나타낼 수 있습니다.

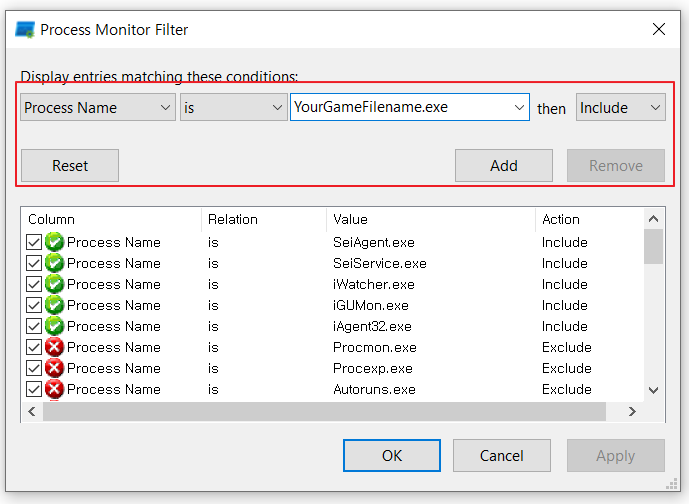
이 섹션에서는 Process Monitor를 사용하여 데이터를 기록하고, 탐색하고, 게임이 상호 작용하는 파일에 대해 정보를 바탕으로 추측하는 방법을 보여드리겠습니다.

**인게임 이벤트 로그 남기기**

Process Monitor의 로그에는 잠재적으로 유용한 모든 정보가 포함될 수 있지만 가장 실용적인 용도는 게임 내 항목 정의와 같은 데이터 파일이 저장될 수 있는 위치를 파악하는 데 도움이 되는 것 입니다. Process Monitor를 시작하면 가장 먼저 표시되는 대화 상자는 Process Monitor Filter(프로세스 모니터 필터)입니다.



이 대화 상자를 사용하면 숫자를 기반으로 이벤트를 표시하거나 억제할 수 있습니다. 모니터링을 시작하려면 Process Name -> is -> 파일이름 -> Include 을 작성하고 Add 버튼을 클릭합니다.



이것은 YourGameFilename.exe에 의해 호출된 이벤트를 표시하도록 Process Monitor에 지시합니다. 적절한 필터를 설정하면 표시된 기본 창으로 이동합니다.

Process Monitor 로그 영역에 표시되는 열을 구성하려면 헤더를 마우스 오른쪽 버튼으로 클릭하고 열 선택을 선택합니다. 인상적인 옵션이 있지만 7가지를 추천합니다.

* Time of Day : 행동이 언제 일어났는지 알 수 있습니다.
* Process Name : 여러 프로세스를 모니터링하지만 일반적으로 게임에 사용되는 단일 프로세스 필터를 사용하는 경우에 유용합니다. 이 옵션을 비활성화하면 소중한 공간을 절약할 수 있습니다.
* Process ID : 프로세스 이름과 비슷하지만 이름이 아닌 ID를 표시합니다.
* Operation : 수행된 작업을 보여줍니다. 따라서 이 옵션은 필수입니다.
* Path : 작업 대상의 경로를 표시합니다. 또한 이 옵션은 필수입니다.
* Detail : 일부 경우에만 유용하지만 활성화해도 문제가 되지 않습니다.
* Result : 파일 로드와 같은 작업이 실패할 때 표시됩니다.

더 많은 열을 표시할수록 로그가 매우 복잡해질 수 있지만 이러한 옵션을 고수하면 출력을 간결하게 유지하는 데 도움이 됩니다.

모니터가 실행 중이고 보고자 하는 열을 정의하면 그림 3-2에 검은색 윤곽선으로 표시된 5개의 이벤트 클래스 필터가 있으며 토글하여 로그를 더 정리할 수 있습니다. 이벤트 클래스 필터를 사용하면 유형에 따라 로그에 표시할 이벤트를 선택할 수 있습니다. 이러한 필터는 왼쪽에서 오른쪽으로 다음과 같습니다.

* File System Activity : 모든 파일 시스템 활동을 표시합니다. 데이터 파일이 저장된 위치와 액세스 방법을 아는 것이 효과적인 봇을 작성하는 데 필수적이기 때문에 이것은 가장 중요한 이벤트 클래스 필터입니다.
* Network Activity : 모든 네트워크 활동을 표시합니다. 네트워크 이벤트에 대한 호출 스택(call stack)은 게임 내에서 네트워크 관련 코드를 찾는데 유용할 수 있습니다.
* Process and Thread Activity : 모든 프로세스 및 스레드 작업을 표시합니다. 이러한 이벤트에 대한 호출 스택(call stack)은 게임 코드가 스레드를 처리하는 방법에 대한 통찰력을 제공할 수 있습니다.
* Profiling Events : 실행 중인 각 프로세스의 메모리 및 CPU 사용량에 대한 정보를 주기적으로 표시합니다. 게임 해커는 거의 사용하지 않습니다.

클래스 수준 이벤트 필터링이 여전히 로그에서 원치 않는 오염을 필터링할 만큼 정확하지 않은 경우 이벤트 수준 필터링 옵션에 대한 특정 이벤트를 마우스 오른쪽 버튼으로 클릭합니다. 필요한 항목만 기록하도록 이벤트 필터링을 구성했으면 로그 탐색을 시작할 수 있습니다.

로그 동작을 제어하는 데 유용한 몇 가지 단축키가 나열되어 있습니다.

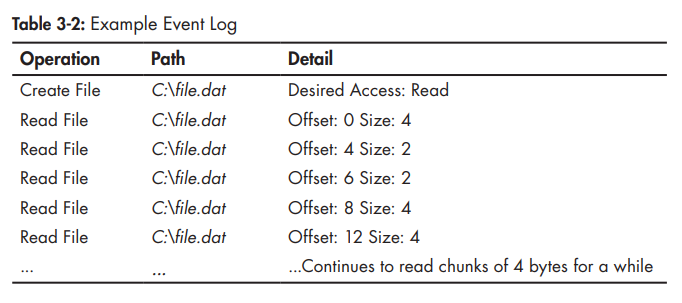
* CTRL-E : 로깅 전환 (캡쳐시작, 캡쳐중지)
* CTRL-A : 로그의 자동 스크롤을 토클합니다. (자동 스크롤이 활성화되어있으면 캡쳐중 자동 스크롤)
* CTRL-X : 로그를 지웁니다.
* CTRL-L : 필터 대화 상자를 표시합니다.
* CTRL-H : 강조 표시 대화상자를 표시합니다. 이 대화 상자는 필터 대화 상자와 매우 유사하지만 강조 표시해야 하는 이벤트를 나타내는 데 사용됩니다.
* CTRL-F : 검색 대화 상자를 표시합니다.
* CTRL-P : 선택한 이벤트에 대한 이벤트 속성 대화 상자를 표시합니다.

로그를 탐색할 때 기록된 작업을 검사하여 이벤트의 세부 정보를 볼 수 있습니다.

**Process Monitor 로그의 이벤트 조사하기**

Process Monitor는 이벤트에 대해 가능한 모든 데이터 포인트를 기록하므로 이벤트가 실행되는 파일뿐만 아니라 이러한 이벤트에 대해 더 많이 알 수 있습니다. Result 및 Detail과 같은 데이터가 풍부한 열을 주의 깊게 검사하면 매우 흥미로운 정보를 얻을 수 있습니다.

예를 들어, 게임이 때때로 파일에서 직접 요소별로 데이터 구조를 읽는다는 것을 발견했습니다. 이 동작은 로그에 동일한 파일에 대한 많은 수의 읽기가 포함되어 있고 각 읽기에는 순차적 오프셋이 있지만 길이가 다른 경우 분명합니다. 아래에 표시된 가상 이벤트 로그를 고려하십시오.



이 로그는 게임이 파일에서 구조를 하나씩 읽고 있으며 구조가 어떻게 생겼는지에 대한 몇 가지 힌트를 보여줍니다. 예를 들어 이러한 읽기가 다음 데이터 파일을 반영한다고 가정해보겠습니다.

struct myDataFile

{

int header; // 4bytes (offset 0)

short effectCount; // 2bytes (offset 4)

short itemCount; // 2bytes (offset 6)

int\* effects;

int\* items;

};

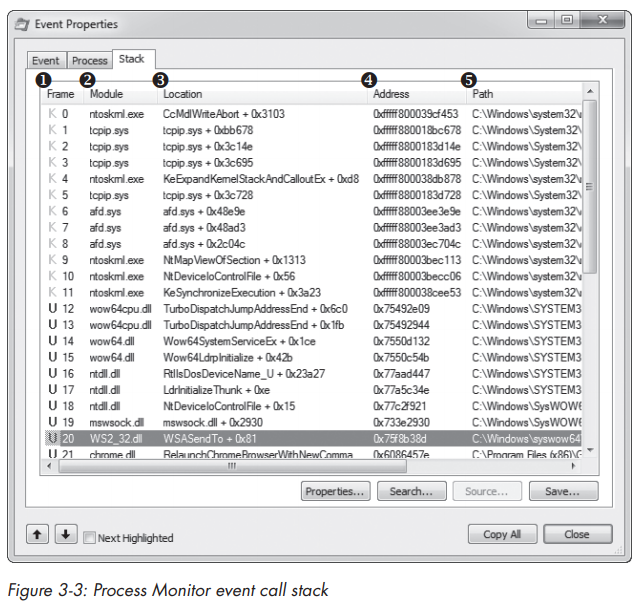
표 3-2의 로그를 이 구조와 비교하십시오. 먼저 게임은 4 헤더 바이트를 읽습니다. 그런 다음 두 개의 2바이트 값인 effectCount 및 itemCount를 읽습니다.

개발자는 확실히 파일에서 데이터를 읽기 위해 이와 같은 프로세스를 사용해서는 안 되지만 이러한 일이 얼마나 자주 발생하는지 보면 놀랄 것 입니다. 다행스럽게도 이와 같은 순진한 기능을 사용하면 분석이 더 쉬워집니다.

이 경우 이벤트 로그는 파일 내의 작은 정보 조각을 식별할 수 있습니다. 그러나 읽기를 알려진 구조와 상관시키는 것은 쉽지만 이벤트 로그 외에는 알려지지 않은 구조를 리버스 엔지니어링 하는 것이 훨씬 더 어렵다는 점을 명심하십시오. 일반적으로 게임 해커는 디버거를 사용하여 흥미로운 각 이벤트에 대한 더 많은 컨텍스트를 얻고 Process Monitor의 데이터를 디버깅 세션에 원활하게 통합하여 두 가지 강력한 리버스 엔지니어링 패러다임을 효과적으로 연결할 수 있습니다.

**더 많은 데이터를 얻기 위해 게임 디버깅하기**

이 가상의 파일 읽기에서 벗어나 Process Monitor를 사용하여 이벤트 로깅에서 디버깅으로 전환하는 방법을 살펴보겠습니다. 프로세스 모니터는 각 이벤트에 대한 전체 스택 추적(stack trace)을 저장하여 이벤트를 트리거한 전체 실행 체인을 보여줍니다.



스택 추적(stack trace)은 실행 모드(execution mode)와 스택 프레임(stack frame) 인덱스를 보여주는 프레임 열로 시작하는 테이블에 표시됩니다. 이 열에서 분홍색 K는 호출이 커널 모드에서 발생했음을 의미하고 파란색 U는 사용자 모드에서 발생했음을 의미합니다. 게임 해커는 일반적으로 사용자 모드에서 작업하기 때문에 커널 모드 작업은 일반적으로 의미가 없습니다.

모듈(Module) 열에는 호출 코드가 있는 실행 가능한 모듈이 표시됩니다. 각 모듈은 호출한 바이너리의 이름일 뿐입니다. 이렇게 하면 게임 바이너리 내에서 실제로 어떤 호출이 발생했는지 쉽게 식별할 수 있습니다.

위치(Location) 열에는 각 호출을 수행한 함수의 이름과 호출 오프셋(call offset)이 표시됩니다. 이러한 함수 이름은 모듈의 내보내기 테이블(Export table)에서 추론되며 일반적으로 게임 바이너리 내의 함수에는 표시되지 않습니다. 함수 이름이 없으면 대신 위치(Location) 열에 모듈 이름(Module name)과 모듈의 기본 주소(Base address)에서 호출의 오프셋(호출이 메모리에 있는 원본 주소를 지난 바이트 수)이 표시됩니다.

주소(Address) 열은 호출의 코드 주소(code address)를 보여주는데, 이는 OllyDbg 디스어셈블러에서 주소(Address)로 점프(jump)할 수 있기 때문에 매우 유용합니다.

마지막으로 Path 열에는 호출한 모듈의 경로가 표시됩니다. 제 생각에 스택 추적(stack trace)은 지금까지 Process Monitor에서 가장 강력한 기능입니다. 이벤트로 이어진 전체 컨텍스트(context)를 보여주므로 게임을 디버깅 할 때 매우 유용할 수 있습니다. 이를 사용하여 이벤트를 트리거한 정확한 코드를 찾고, 호출 체인을 크롤링하여 이벤트가 발생한 경로를 확인하고, 각 작업을 완료하는데 사용된 라이브러리를 정확히 결정할 수도 있습니다.

**Finding a High Score File**





Process Monitor의 자매 응용프로그램인 Process Explorer에는 Process Monitor 또는 OllyDbg 이상의 기능이 없습니다. 그러나 이러한 기능 중 일부를 훨씬 더 효과적으로 노출하므로 특정 상황에서 이상적인 선택이 됩니다.

**Process Explorer**

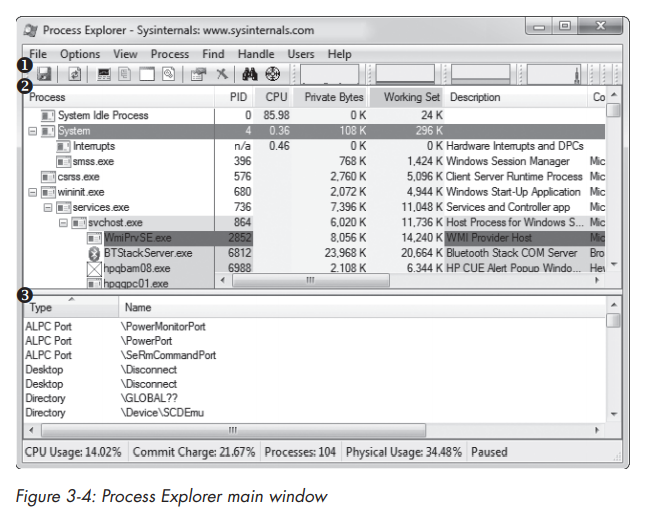
Process Explorer는 고급 작업 관리자(기본 작업 관리자로 만들기 위해 누를 수 있는 버튼도 있음)이며 게임 작동 방식을 이해하기 시작할 때 매우 편리합니다. 상위(parent) 및 하위(child) 프로세스, CPU 사용량, 메모리 사용량, 로드된 모듈(loaded modules), 열린 핸들(open handles) 및 명령줄 인수(command line arguments)와 같은 실행 중인 프로세스에 대한 복잡한 데이터를 제공하고 이러한 프로세스를 조작할 수 있습니다. 프로세스 트리, 메모리 소비, 파일 액세스 및 프로세스 ID와 같은 높은 수준의 정보를 표시하는 것 보다 훨씬 더 유용할 수 있습니다.

물론 이 데이터 중 어떤 것도 따로 유용하지 않습니다. 그러나 예리한 눈으로 상관 관계를 만들고 파일(Files), 뮤텍스(Mutexes), 공유 메모리 세그먼트(Shared memory segments)를 포함한 전역 객체(Global objects)에 대해 게임이 액세스 할 수 있는 유용한 결론을 도출할 수 있습니다. 또한 Process Explorer에 표시된 데이터는 디버깅 세션에서 수집 된 데이터와 상호 참조될 때 훨씬 더 가치가 있을 수 있습니다.

이 섹션에서는 Process Explorer 인터페이스를 소개하고 표시되는 속성에 대해 설명하며 이 도구를 사용하여 핸들(시스템 리소스에 대한 참조)을 조작하는 방법을 설명합니다. 이 소개 후에 60페이지의 “뮤텍스 찾기 및 닫기(Finding and Closing Mutex)”기를 사용하여 기술을 연마하십시오.

**Process Explorer의 사용자 인터페이스와 제어**

Process Explorer를 열면 그림 3-4와 같이 세 개의 개별 섹션으로 분할된 창이 표시됩니다.



이 세 섹션은 도구 모음(toolbar), 위쪽 창(upper pane)에는 상위/하위(parent/child) 관계를 표시하기 위해 트리 구조를 사용하는 프로세스 목록이 표시됩니다. 다른 프로세스는 다른 색상으로 강조 표시됩니다. 현재 색상이 마음에 들지 않으면 Options -> Configure Colors(색상 구성)을 클릭하여 색상을 보고 변경할 수 있는 대화 상자를 표시합니다.

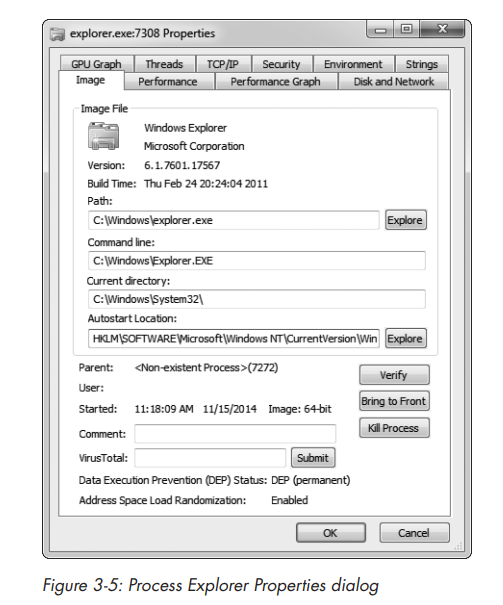
Process Monitor에서와 마찬가지로 이 테이블의 디스플레이는 매우 다양하며 테이블 헤더를 마우스 오른쪽 버튼으로 클릭하고 열 선택을 하여 사용자 정의할 수 있습니다. 100개 이상의 사용자 지정 옵션이 있을 수 있지만 ASLR 사용 열을 추가한 기본값이 제대로 작동한다는 것을 알았습니다.

ASLR은 예측할 수 없는 위치에 실행 가능한 이미지를 할당하는 보안 기능이며 메모리의 게임 상태 값을 변경하려고 할 때 켜져 있는지 여부를 아는 것은 매우 중요합니다.

아래쪽 창(lower pane)에는 숨김(Hidden), DLLs 및 핸들(Handles)의 세 가지 가능한 상태가 있습니다. Hidden 옵션은 창을 보기에서 숨기고 DLL은 현재 프로세스 내에서 로드된 Dynamic Link Libraries 목록을 표시하며 Handles는 프로세스가 보유한 핸들 목록을 표시합니다.

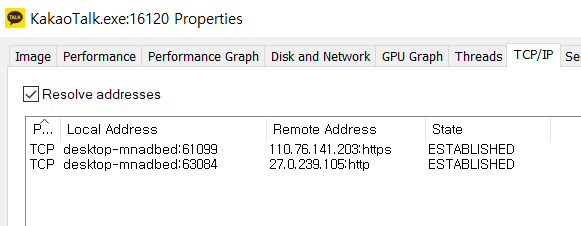
**프로세스 속성 조사하기**

Process Monitor와 마찬가지로 Process Explorer는 데이터 수집에 대해 매우 동적인 접근 방식을 사용합니다. 최종 결과는 정보의 광범위하고 장황한 스펙트럼입니다. 실제로 프로세스에 대한 속성 대화 상자(그림3-5)를 열면 10개의 탭이 포함된 거대한 탭 막대가 표시됩니다.

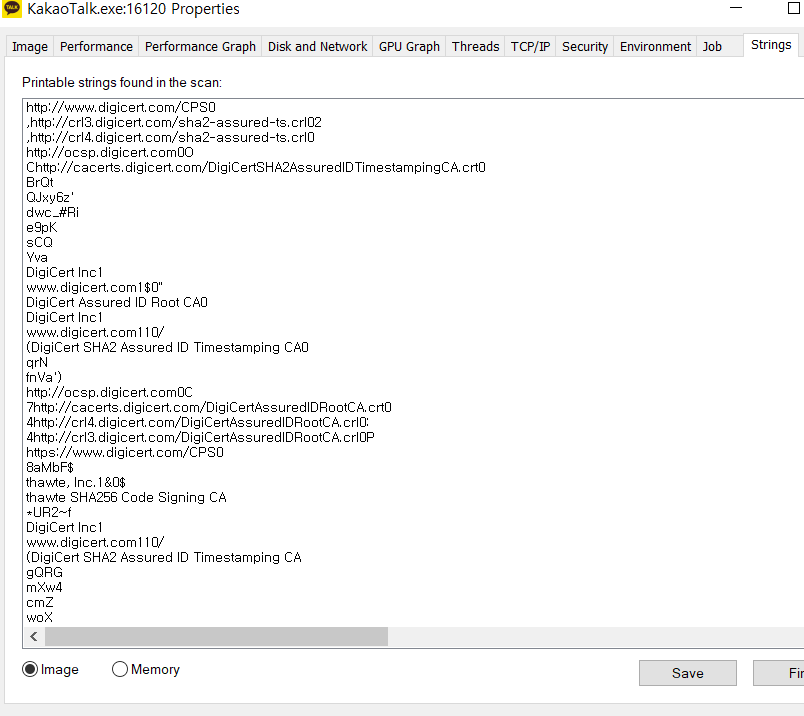


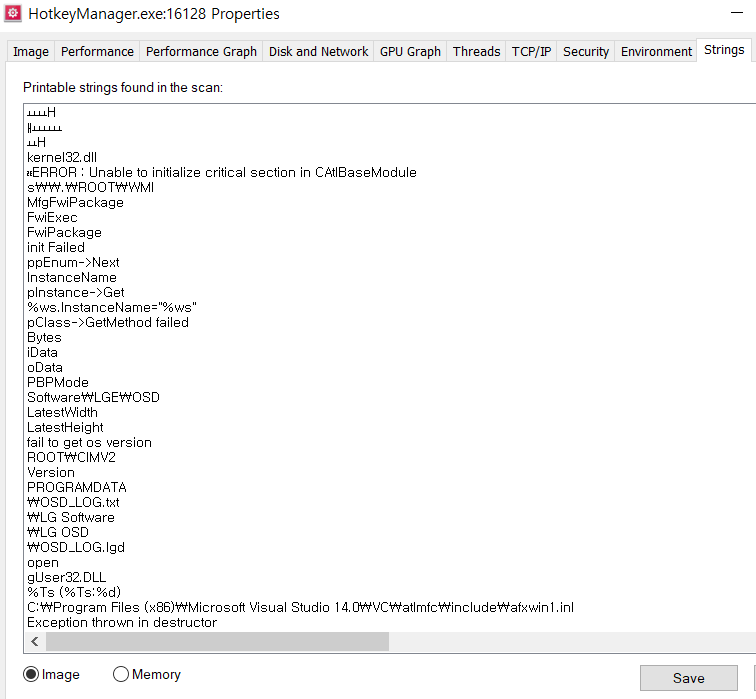
성능(Performance), 성능 그래프(Performance Graph), 디스크 및 네트워크(Disk and Network), GPU 그래프(GPU Graph) 탭에는 프로세스의 GPU, 메모리(Memory), 디스크(Disk), 네트워크(Network) 및 GPU 사용량에 대한 수많은 메트릭이 표시됩니다. 게임에 주입하는 봇을 만드는 경우 이 정보는 봇이 게임에 미치는 성능 영향을 결정하는 데 매우 유용할 수 있습니다.

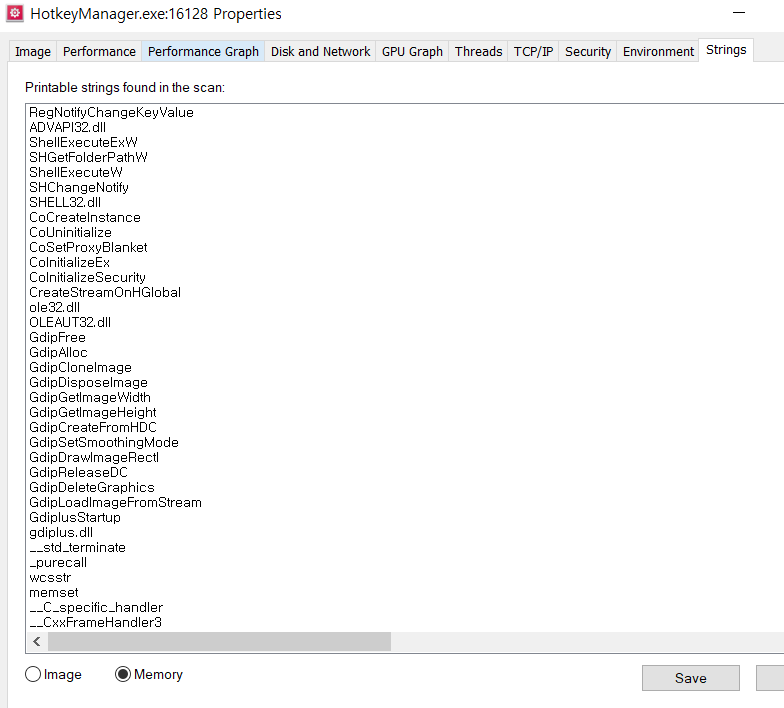
TCP/IP 탭에는 게임이 연결되는 모든 게임 서버 IP 주소를 찾는 데 사용할 수 있는 활성 TCP 연결 목록이 표시됩니다. 연결 속도를 테스트하거나 연결을 종료하거나 게임의 네트워크 프로토콜을 조사하려는 경우 이 정보가 중요합니다.



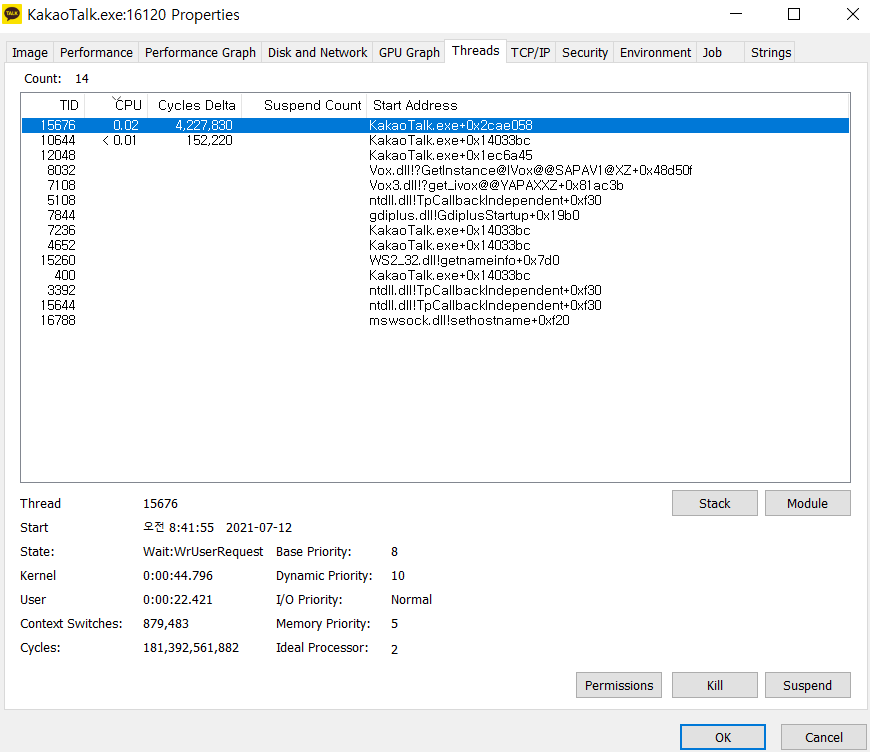
문자열(Strings) 탭은 바이너리 또는 프로세스의 메모리에서 찾은 문자열 목록을 표시합니다. 어셈블리 코드에서 참조하는 문자열만 표시하는 OllyDbg의 문자열 목록과 달리 목록에는 3개 이상의 연속 가독성 문자와 그 뒤에 null 종결자가 나오는 모든 항목이 포함됩니다. 게임 바이너리가 업데이트되면 각 게임 버전에서 이 목록의 diffing 도구를 사용하여 조사하려는 새 문자열이 있는지 확인할 수 있습니다.

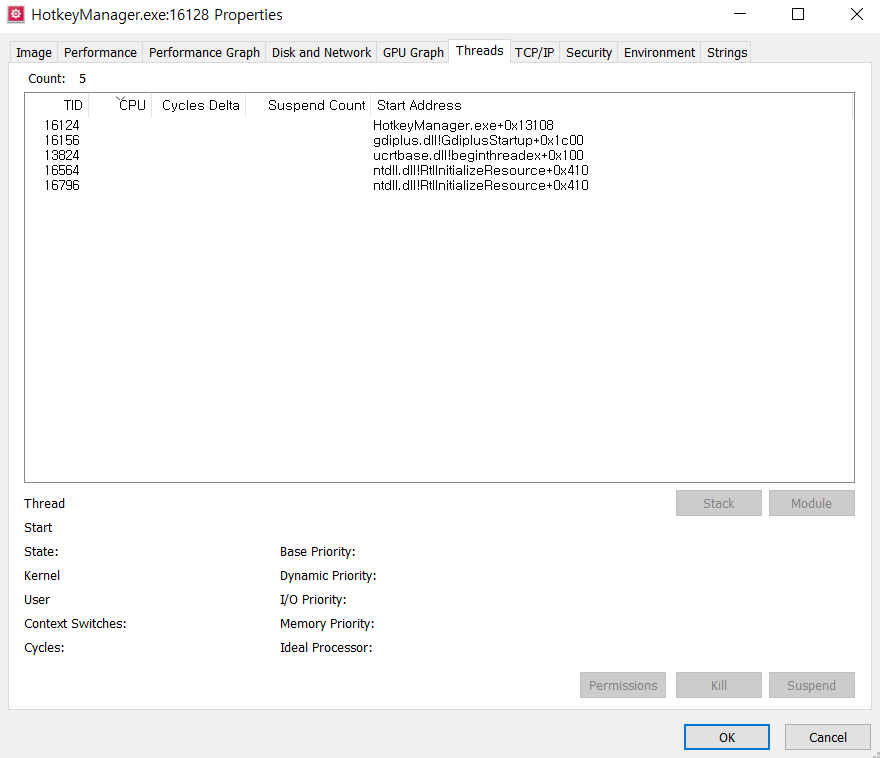




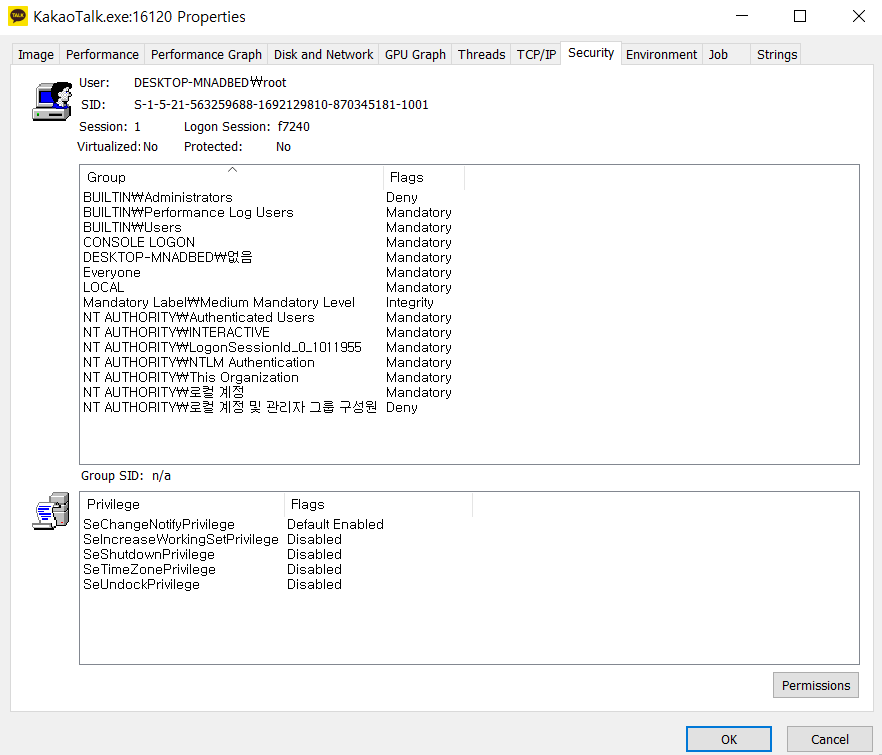


스레드(Threads) 탭은 프로세스 내에서 실행 중인 스레드 목록을 표시하며 각 스레드를 일시 중지, 재개 또는 종료할 수 있습니다.

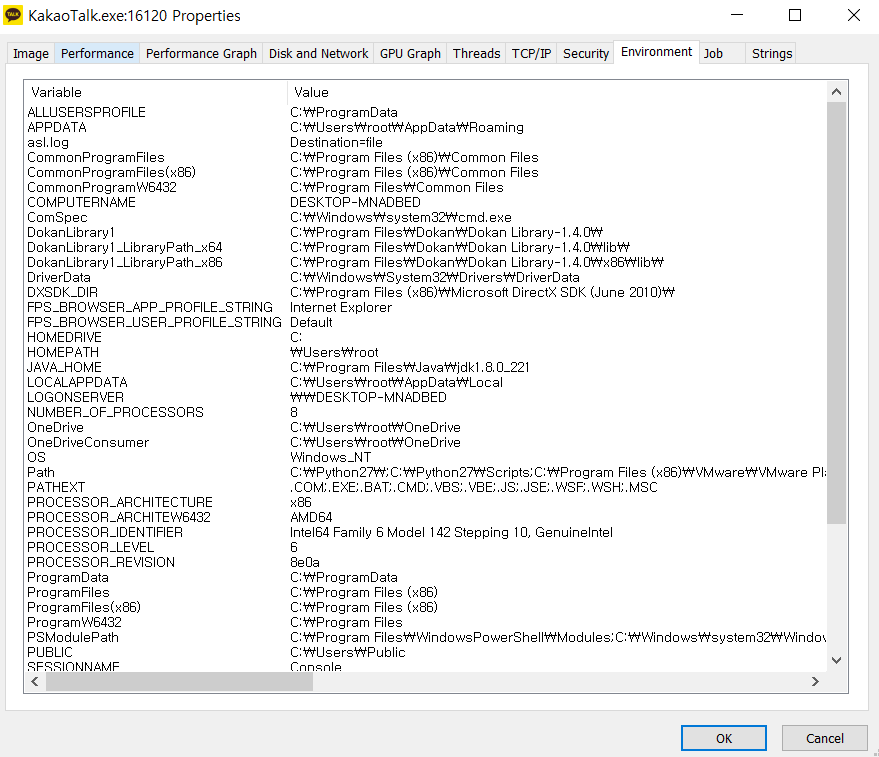




보안(Security) 탭은 프로세스의 보안 권한을 표시합니다.



환경(Environment) 탭에는 프로세스에 알려지거나 프로세스에 의해 설정된 모든 환경 변수가 표시됩니다.



.NET 프로세스의 속성 대화상자를 열면 .NET 어셈블리(.NET Assemblies)와 .NET 성능(.NET Performance)이라는 두 개의 추가 탭이 표시됩니다. 이 탭의 데이터는 매우 자명합니다.

**핸들 조작 옵션**

지금까지 살펴본 것처럼 Process Explorer는 프로세스에 대한 풍부한 정보를 제공할 수 있습니다. 하지만 이것이 좋은 것은 아닙니다. 프로세스의 특정 부분을 조작할 수도 있습니다. 예를 들어 Process Explorer의 아래쪽 창(lower pane)에서 편안하게 열린 핸들을 보고 조작할 수 있습니다. 이것만으로도 도구 상자에 Process Explorer를 추가해야 하는 강력한 주장이 됩니다. 핸들을 닫는 것은 핸들을 마우스 오른쪽 버튼으로 클릭하고 핸들 닫기를 선택하는 것 만큼 간단합니다. 이것은 예를 들어 특정 유형의 해킹에 필수적인 뮤텍스를 닫고 싶을 때 유용할 수 있습니다.

아래쪽 창(lower pane) 머리글(header)을 마우스 오른쪽 버튼으로 클릭하고 열 선택(Select Columns)을 클릭하여 표시를 사용자 지정할 수 있습니다. 특히 유용할 수 있는 열 중 하나는 핸들 값으로, OllyDbg에서 핸들이 전달되는 것을 보고 핸들이 무엇을 하는지 알고싶을 때 도움이 될 수 있습니다.

**Closing Mutexes(뮤텍스 닫기)**

게임에서는 종종 한 번에 하나의 클라이언트만 실행할 수 있습니다. 이것을 단일 인스턴스 제한(single instance limitation)이라고 합니다. 여러 가지 방법으로 단일 인스턴스 제한을 구현할 수 있지만 뮤텍스(Mutex)는 세션 전체에 걸쳐 있고 간단한 이름으로 액세스 할 수 있기 때문에 시스템 뮤텍스(System Mutex)를 사용하는 것이 일반적입니다. 뮤텍스로 인스턴스를 제한하는 것은 간단하며 Process Explorer 덕분에 이 제한을 제거하는 것도 간단하여 게임의 여러 인스턴스를 동시에 실행할 수 있습니다.

먼저 게임이 뮤텍스(Mutex)를 사용하여 단일 인스턴스 제한(single instance limitation)을 하는 방법은 다음과 같습니다:

int main(int argc, char \*argv[])

{

// create the mutex

HANDLE mutex = CreateMutex(NULL, FALSE, “onlyoneplease”);

if (GetLastError() == ERROR\_ALREADY\_EXISTS) {

// the mutex already exists, so exit

ErrorBox(“An instance is already running.”);

return 0;

}

// the mutex didn’t exist; it was just created, so

// let the game run

RunGame();

// the game is over; close the mutex to free it up

// for future instances

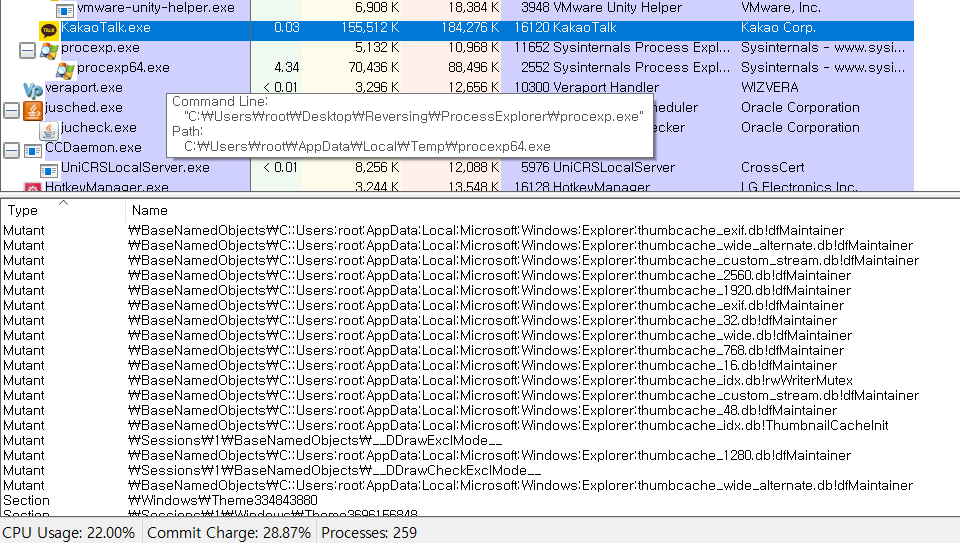
if (mutex)

CloseHandle(mutex);

}

이 예제 코드는 onlyoneplease라는 뮤텍스(Mutex)를 만듭니다. 다음으로, 함수는 GetLastError()를 확인하여 뮤텍스(Mutex)가 이미 생성되었는지 확인하고, 생성된 경우 게임을 닫습니다. 뮤텍스(Mutex)가 아직 존재하지 않는 경우 게임은 첫 번째 인스턴스를 생성하여 향후 게임 클라이언트가 실행되는 것을 차단합니다. 이 예에서 게임은 정상적으로 실행되고 일단 완료되면 CloseHandle()이 호출되어 뮤텍스(Mutex)를 닫고 향후 게임 인스턴스를 실행할 수 있습니다.

Process Explorer를 사용하여 인스턴스 제한 뮤텍스를 닫고 많은 게임 인스턴스를 동시에 실행할 수 있습니다. 이렇게 하려면 아래쪽 창(lower pane)의 핸들 보기를 선택하고 Mutant 유형의 모든 핸들을 찾아 게임 인스턴스를 제한하는 핸들을 결정하고 해당 뮤텍스를 닫습니다.

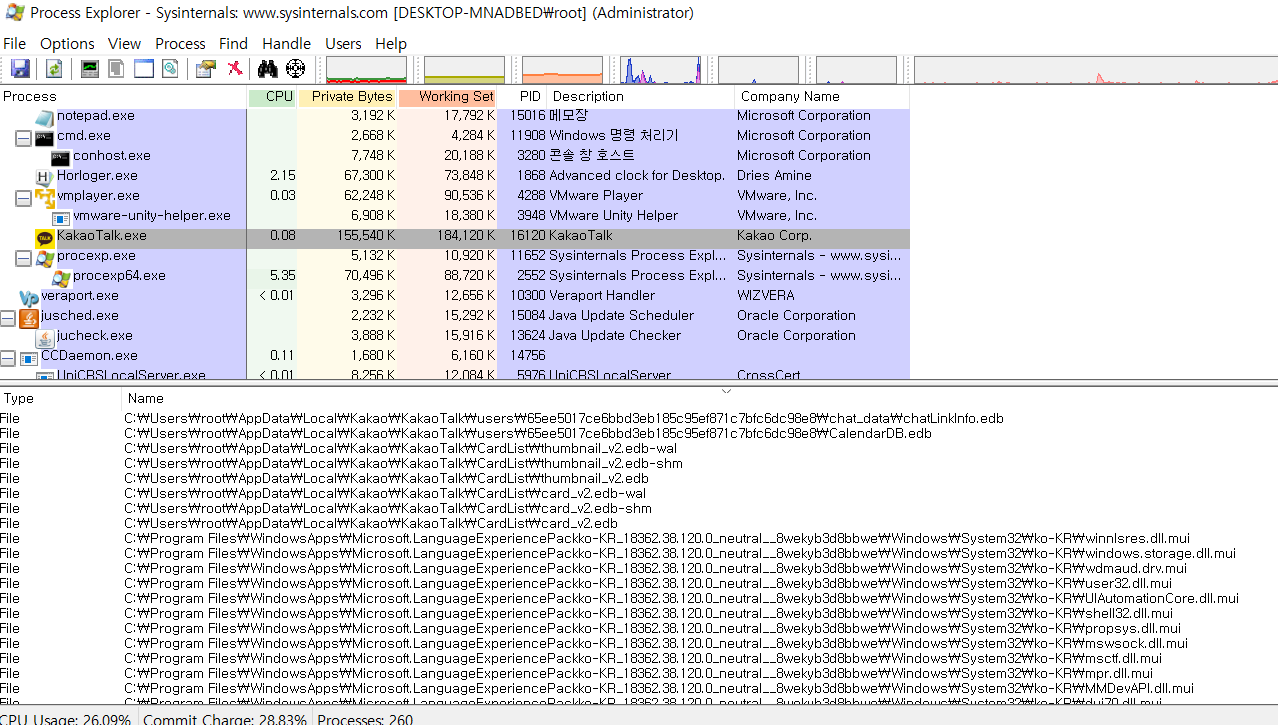


뮤텍스(Mutex)는 스레드(Thread)와 프로세스(Process) 간에 데이터를 동기화(synchronize)하는 데에도 사용됩니다. 그것의 유일한 목적이 당신이 전복시키려는 것이라고 확신하는 경우에만 하나를 닫으십시오!

멀티클라이언트 해킹은 일반적으로 수요가 높기 때문에 신흥 게임을 위해 빠르게 개발할 수 있는 능력은 해당 시장에서 봇 개발자로서 전반적인 성공에 매우 중요합니다. 뮤텍스(Mutex)는 단일 인스턴스 제한(Single instance limitation)을 달성하는 가장 일반적인 방법 중 하나이기 때문에 Process Explorer는 이러한 종류의 핵을 프로토타이핑하기 위한 필수 도구입니다.

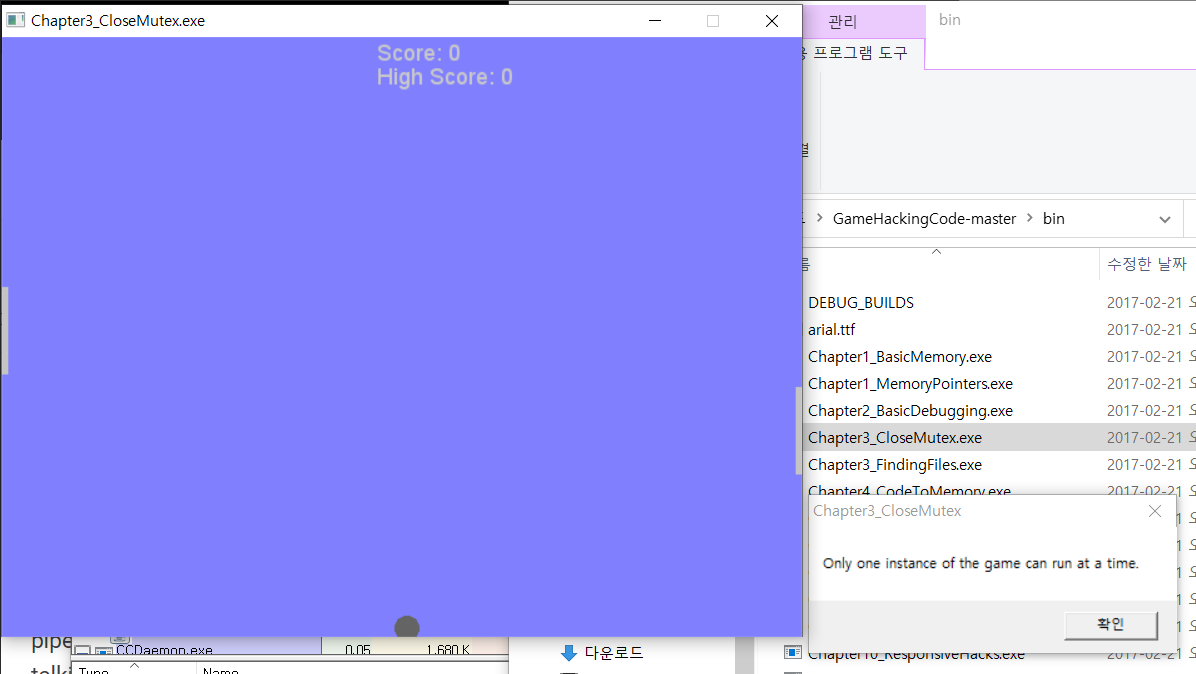
**Inspecting File Accesses(파일 액세스 검사)**

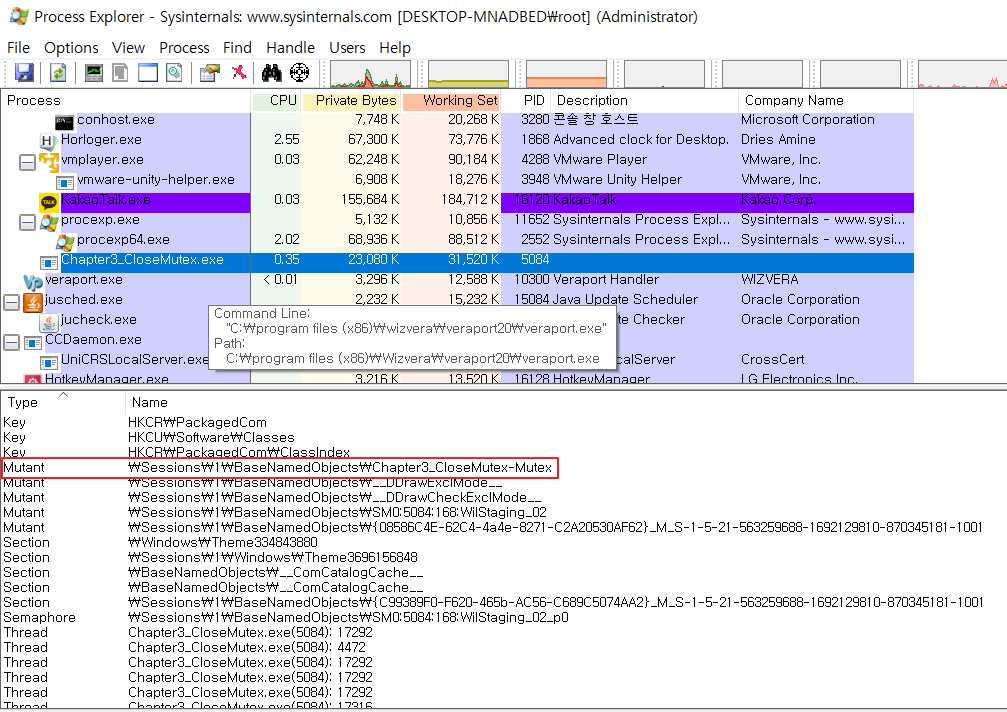
Process Monitor와 달리 Process Explorer는 파일 시스템 호출(filesystem call) 목록을 표시할 수 없습니다. 반면에 Process Explorer의 아래쪽 창(lower pane)에 있는 핸들 보기는 현재 게임에서 열려 있는 모든 파일 핸들을 표시할 수 있으므로 Process Monitor에서 고급 필터링 기준을 설정할 필요 없이 어떤 파일이 지속적으로 사용되는지 정확히 알 수 있습니다. 게임에서 현재 사용 중인 모든 파일을 보려면 파일 유형의 핸들을 찾으십시오.

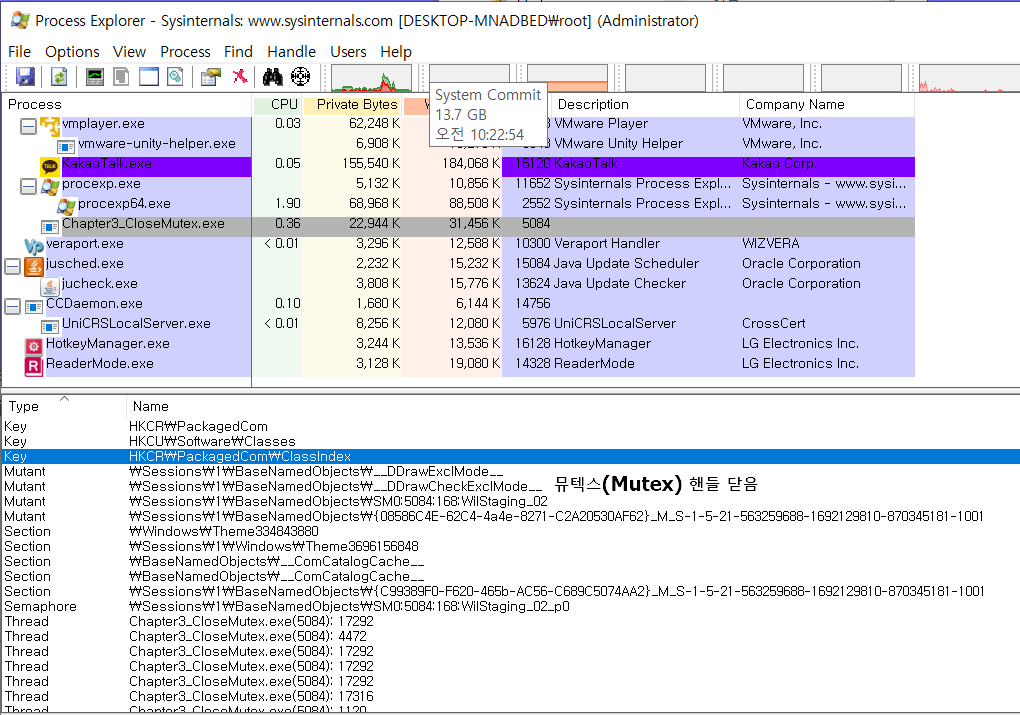


이 기능은 로그 파일을 찾거나 파일을 저장하려는 경우에 유용할 수 있습니다. 또한 IPC(프로세스 간 통신)에 사용되는 명명된 파이프(named pipes)를 찾을 수 있습니다. \Device\NamedPipe\ 접두사가 붙은 파일입니다. 이러한 파이프 중 하나를 보는 것은 종종 게임이 다른 프로세스와 대화하고 있다는 암시입니다.

**Finding and Closing a Mutex**









* Single instance limitation mutex의 handle을 닫은 후 멀티클라이언트가 실행 가능