다시 시작하는 윈도우 프로그래밍
메모리 이야기 (2)

목차

목차 1
소개 1
연재 가이드 1
필자소개 1
필자 메모 2
Introduction 2
가상 메모리의 기본 개념 3
가상 메모리 할당 4
가상 메모리 해제 7
할당과 해제 7
가상 메모리 상태 조회 및 보호 속성 변경 9
vmwalk 11
참고자료 13

소개

윈도우 메모리 관리의 내부 밑 단계에는 가상 메모리 관리자가 존재한다. 지난 시간에 살펴보았던 힙과 스택도 이러한 가상 메모리 관리자의 도움을 받아 구현되는 것들이다. 이번 시간에는 이러한 가상 메모리의 개념에 대해서 살펴보고, 가상 메모리를 조작하는 API의 사용법에 대해서 알아본다.

연재 가이드

운영체제: Windows XP
개발도구: Visual Studio 2005
기초지식: C/C++ 문법
응용분야: 윈도우 응용 프로그램

필자소개

신영진 pop@jiniya.net, http://www.jiniya.net

웹비아닷컴에서 보안 프로그래머로 일하고 있다. 시스템 프로그래밍에 관심이 많으며 다수의 PC 보안 프로그램 개발에 참여했다. 현재 데브피아 Visual C++ 섹션 시삽과 Microsoft Visual C++ MVP로 활동하고 있다. C와 C++, Programming에 관한 이야기를 좋아한다.

Copyright © 2011, 신영진

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

필자 메모

벌써 여름이다. 날씨가 더워지면서 새로 이사한 집의 에어컨을 가동할 일이 있었다. 첫 입주라 처음 리모컨 비닐을 뜯고 에어컨을 동작시키는 영광을 얻을 수 있었다. 에어컨은 가동했고, 방은 아주 시원해졌다. 한 시간 정도 생각 없이 놀다가 추워서 에어컨을 끄러 갔다. 그런데 왠 일인지 에어컨 밑에 물이 심하게 떨어지고 있는 것이 아닌가? 삼십 분만 늦게 봤으면 집이 물바다가 될 뻔한 상황이었다.

일단 에어컨을 끄고 왜 그 곳으로 물이 떨어졌는지를 관찰했다. 화장실 쪽으로 연결해둔 배수구가 문제였다. 배수구로 물이 정상적으로 배출이 되지 않아 에어컨은 토를 해대고 있었던 것이다. 길게 늘어진 배출구를 두드리자 미친 듯이 물을 내뿜기 시작했다.

처음이라 그랬겠지 라는 생각에 다시 에어컨을 가동 시켰다. 다시 삼십 분 정도의 시간이 흘렀다. 에어컨은 다시 토를 하기 시작했다. 난 다시 배출구로 갔고 배출구가 검은 전기 테이프로 막혀 있는 것을 발견했다. 그것을 뜯고 두드리자 다시 물이 배출구로 나오기 시작했다.

이젠 고쳐졌겠지 라고 생각하며 다시 에어컨을 가동 시켰다. 다시 삼십 분 후 에어컨은 여전히 토를 하기 시작한다. 난 다시 배출구를 살핀다. 중요한 결함을 이제서야 눈치챈다. 쭉 늘어진 배출구 끝이 올라가 있어서 물이 정상적으로 빠져 나오지 못하는 것이었다. 즉 배출구를 절단해야 하는 상황이었다. 고민이 앞선다. 배출구를 절단했다가 더 큰 문제가 생기는 것은 아닐까? 하지만 이내 배출구를 절단 했다. 그 날 이 후 에어컨은 토를 하지 않았고, 시원하게 잘 동작한다.

재미 없는 에어컨 이야기를 이리도 길게 쓴 이유는 나는 저 이야기에서 많은 것을 느낄 수 있었기 때문이었다. 에어컨이 토를 하는 상황은 버그였고, 나는 그것을 고치는 개발자였다고 생각해보자. 나는 왜 그 버그를 한 번에 고치지 못했을까? 저 문제를 한 번에 해결하기 위해서는 어떠한 식으로 문제에 접근해야 했을까? 또는 에어컨이 토를 하는 버그가 발생하기 전에 문제점을 발견해 낼 수 있는 방법은 없었을까?

Introduction

멀티태스킹은 정말 마법 같은 일이다. 어떻게 그렇게 많은 프로그램들이 동시에 동작할 수 있을까? 물론 동시에 동작한다는 사실만이 놀라운 것은 아니다. 제한된 물리 메모리 내에서 그렇게 많은 프로그램들을 동시에 실행한다는 것은 더 신기한 일이다. 물론 그 마법 뒤에서 벌어지는 일을 알고 있다면 전혀 신기한 일이 아니지만 말이다.

우리는 지난 시간에 스택과 힙 메모리에 대해서 살펴보았다. <그림 1>에 나와 있는 것처럼 스택과 힙 메모리는 윈도우 메모리 관리의 상단에 위치한 것으로 가상 메모리 API의 도움을 얻어 구현된다. 이번 시간에는 앞서 소개한 마법 같은 일의 핵심적인 기능을 하는 가상 메모리의 개념과

Copyright © 2011, 신영진
이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

그것을 사용하는 API에 대해서 알아볼 것이다. 끝으로 해당 지식을 사용해서 vmwalk라는 간단한 프로그램을 제작해 본다.

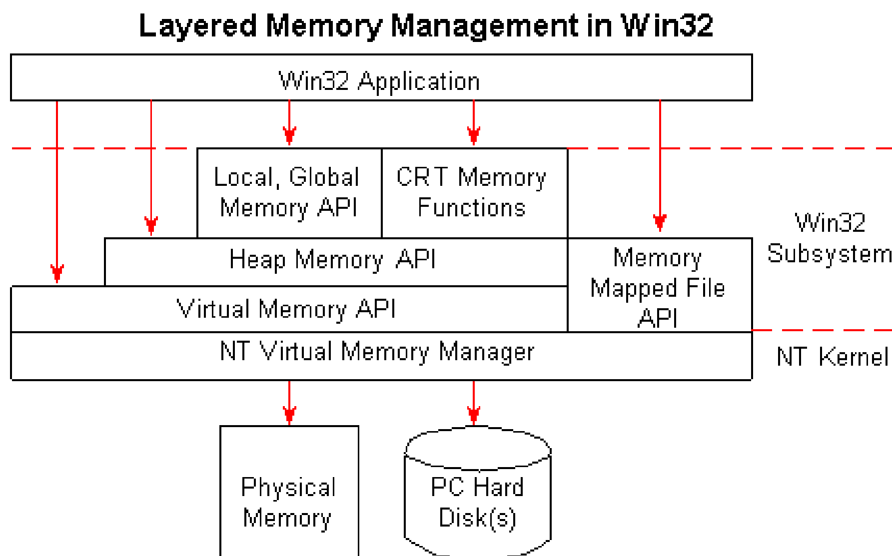

그림 1 윈도우 메모리 아키텍처

가상 메모리의 기본 개념

가상 메모리의 개념을 이해하기 위한 첫 단추는 페이징과 스와핑이다. 페이징이란 물리 메모리 주소와 논리 메모리 주소를 분리 시키는 것을 의미한다. 운영체제에서 사용되는 0x100이란 주소가 실제 물리적인 메모리의 0x100번지를 의미하지 않는다는 이야기다. 스와핑이란 이렇게 분리된 물리 주소 공간을 확장 시키는 기능으로 메모리에서 현재 참조되지 않는 부분을 디스크에 저장하는 기술을 말한다. 물론 해당 메모리가 다시 참조되는 순간 디스크에서 다시 불려진다.

페이징은 페이지 테이블이라는 것을 통해서 구현된다. 페이지 테이블은 논리 주소를 실제 물리 주소로 맵핑시키는 테이블이다. 즉 0x12345678이라는 주소 번지가 실제 물리 메모리의 어떤 위치에 기록되었는지를 찾기 위해서 CPU는 페이지 테이블 참조해서 실제 메모리 번지를 알아내고, 그 번지를 접근해서 데이터를 읽는 것이다. 이렇게 함으로써 실제로 컴퓨터에 탑재된 물리 메모리의 양에 상관없이 더 넓은 주소 공간을 사용하도록 할 수 있다. 또한 페이징을 사용하는 운영체제는 응용 프로그램에게 실제 그 주소가 어떤 물리 메모리에 맵핑되었는지를 감춤으로 해서 높은 수준의 추상화와 보안성을 가지게 된다.

윈도우는 이러한 페이징 기술을 사용해서 시스템의 물리 메모리의 양에 관계없이 32비트 컴퓨터라면 4GB의 논리적인 주소 공간을 제공해 준다. 여기다 윈도우는 앞서 설명한 페이지 테이블을 프로세스 별로 관리해서 각각의 프로세스 별로 별도의 메모리 공간을 제공해 준다. 이렇게 분리된 주소 공간을 윈도우에서는 가상 메모리라 부른다. 즉, 실제 메모리가 아닌 가상의 논리 메모리란 의미로 이해하면 되겠다.

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

가상 메모리는 해제(FREE), 예약(RESERVE), 할당(COMMIT) 이라는 세 가지 상태로 관리된다. 해제 상태는 해당 가상 메모리에 접근을 할 수 없는 단계이다. 이 상태의 메모리는 페이지 테이블이 존재하지 않는다. 예약 단계는 해당 가상 메모리에 대한 페이지 테이블이 생성된 단계이다. 하지만 실제 물리 메모리가 해당 페이지에 연결되지는 않은 단계다. 끝으로 할당 단계는 해당 가상 메모리 주소를 사용할 수 있으며, 물리 메모리가 연결된 단계다.

박스 1 유저 메모리와 커널 메모리

윈도우는 커널과 응용 프로그램을 위한 메모리 공간을 분리해서 사용한다. 커널 모드에서 동작하는 프로그램은 모든 메모리 영역이 참조 가능하지만, 일반 응용 프로그램은 커널 메모리에 접근할 수 없다. 이렇게 설정한 이유는 시스템의 안정성을 높이기 위해서다. 응용 프로그램이 커널 주요부의 메모리를 접근하지 못하게 만듦으로써 시스템 내부 상태가 쉽게 변경되는 실수를 막을 수 있기 때문이다.

<그림 2>에는 일반적으로 윈도우의 메모리 영역이 분리된 것이 나와있다. 보통의 경우 윈도우는 왼쪽 그림과 같이 커널을 위해 상위 2GB의 메모리를 사용하며, 응용 프로그램을 위해 하위 2GB의 메모리를 사용한다. 오른쪽 그림은 /3GB 옵션을 사용해 부팅한 경우로 이 경우에 커널 메모리는 1GB로 줄어들고, 유저 메모리는 3GB로 늘어난다.

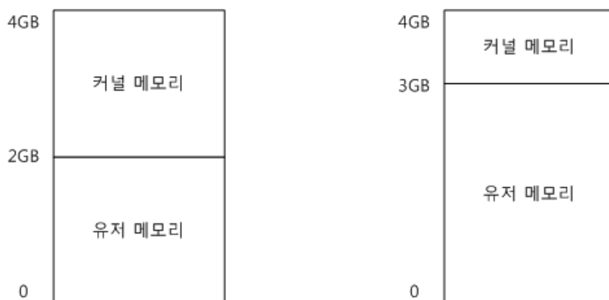

그림 2 윈도우 메모리 구조

가상 메모리 할당

가상 메모리 함수는 크게 두 가지 군으로 나뉜다. 일반 함수와 Ex 함수가 그것이다. Ex 함수는 다른 프로세스의 메모리를 조작할 수 있다는 추가적인 점 외에는 일반 함수와 기능면에서 100% 동일하다. 여기서는 일반 함수의 사용법에 대해서만 살펴본다.

가상 메모리를 할당하기 위해서 윈도우는 VirtualAlloc, VirtualAllocEx라는 두 가지 API를 제공한다. VirtualAlloc은 현재 프로세스의 주소 공간에 메모리를 할당하는 역할을 한다. 앞서 설명한 것과 같이 VirtualAllocEx 함수는 특정 프로세스의 주소 공간에 메모리를 할당하는 기능을 추가적으로

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

가지고 있다. 이 함수의 원형은 다음과 같다.

LPVOID VirtualAlloc(LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType, DWORD flProtect);

첫 번째 인자인 lpAddress는 할당할 베이스 주소 번지를 지정한다. NULL을 지정하면 시스템이 알아서 비어있는 공간을 찾아서 할당한다. lpAddress는 예약(MEM_RESERVE) 작업을 수행하는 경우에는 내부적으로 페이지 할당 단위의 배수로 조정된다. 또한 할당(MEM_COMMIT) 작업을 수행하는 경우에는 페이지 단위의 배수로 조정된다.

두 번째 인자인 dwSize에는 할당 하려고 하는 메모리 크기를 지정한다. lpAddress가 NULL인 경우에 VirtualAlloc함수는 dwSize를 페이지 크기의 배수배가 되도록 조정한다. lpAddress가 NULL이 아닌 경우에는 해당 크기 영역을 포함하는 모든 페이지가 포함되도록 조정된다.

fAllocationTYpe에는 할당 하려고 하는 타입을 지정한다 (<표 1> 참고). 예약과 할당을 한번에 하기 위해서는 MEM_COMMIT | MEM_RESERVE와 같이 지정할 수 있다.

표 1 fAllocationType 값의 의미

값	의미
MEM_COMMIT	메모리를 할당한다.
MEM_RESERVE	메모리를 예약한다.
MEM_RESET	해당 페이지들을 현재 사용하지 않는다는 것을 운영체제에 알려준다. 해당 페이지들은 스왑되어서 페이지 파일에 기록된다.

끝으로 fProtect에는 페이지 보호 속성을 지정한다. fProtect에는 <표 2>에 나와 있는 값 중 하나를 지정한다. 여기에 추가적으로 <표 3>에 나와 있는 값을 지정할 수 있다.

표 2 fProtect 값의 의미

값	의미
PAGE_EXECUTE	이 페이지는 실행 가능하다. 해당 페이지에 읽기, 쓰기 접근을 시도하는 경우 잘못된 접근 오류가 발생한다.
PAGE_EXECUTE_READ	이 페이지는 실행, 읽기 작업이 가능하다.
PAGE_EXECUTE_READWRITE	이 페이지는 실행, 읽기, 쓰기 작업이 가능하다.
PAGE_EXECUTE_WRITECOPY	이 페이지는 실행, 읽기, 쓰기 작업이 가능하다. 추가적으로 기록시 복사 기능을 가진다. VirtualAlloc, VirtualAllocEx, CreateFileMapping 함수는 이 플래그를 지원하지 않는다.
PAGE_NOACCESS	이 페이지는 어떠한 작업도 허용되지 않는다. CreateFileMapping 함

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

	수는 이 플래그를 지원하지 않는다.
PAGE_READONLY	이 페이지는 읽기 작업만 가능하다.
PAGE_READWRITE	이 페이지는 읽기, 쓰기 작업이 가능하다.
PAGE_WRITECOPY	이 페이지는 기록 시 복사 기능을 가진다. VirtualAlloc, VirtualAllocEx 함수는 이 플래그를 지원하지 않는다.

표 3 fProtect에 추가적으로 지정할 수 있는 값의 의미

값	의미
PAGE_GUARD	이 페이지는 접근 시에 STATUS_GUARD_PAGE_VIOLATION 예외가 발생한다.
PAGE_NOCACHE	이 페이지는 캐싱되지 않는다.
PAGE_WRITECOMBINE	이 페이지는 쓰기 조합 최적화 기능을 사용할 수 있다.

간단하게 함수 사용 방법에 대해서 살펴보았다. 할당과 해제를 하는 실질적인 코드는 나중에 자세히 살펴보도록 하자.

박스 2 기록시 복사(Copy on write)

게으른 최적화 기법의 대표적인 방법으로 동일한 내용을 참조하는 경우에 쓰기가 발생하는 경우에 해당 메모리를 로컬 메모리로 복사하는 것을 의미한다. A와 B라는 프로세스가 모두 kernel32.dll을 사용한다고 가정해보자. 두 프로세스가 로딩되는 시점에는 메모리에는 kernel32.dll이 하나만 올라와 있고, 두 프로세스 모두 동일한 영역을 참조한다. A라는 프로세스가 kernel32.dll의 메모리를 조작하면, 운영체제는 해당 메모리를 A프로세스 영역에 복사한 다음 변경한다. 일반적으로 이미지의 코드 내용 자체가 변경되는 경우는 거의 없기 때문에 이러한 방식의 최적화 방법은 매우 효과적이다.

박스 3 쓰기 조합 최적화(Write combine)

쓰기 조합 최적화란 여러 번의 쓰기 작업을 운영체제가 최적화 시키는 것을 말한다. 예를 들어, A라는 프로세스가 100번지에 'A'라는 값을 쓰고, 또 'B'라는 값을 기록하는 두 번의 호출을 했다면, 운영체제는 두 번째 호출만 실제적으로 수행하는 것을 말한다.

박스 4 GetSystemInfo

VirtualAlloc 함수 설명을 보면 페이지 크기라는 말과 할당 단위라는 말이 나온다. 페이지 크기라는 말은 하나의 페이지 크기를 나타낸다. 4GB의 공간을 바이트 단위로 페이지 테이블을 만든다면 4GB보다 더 큰 메모리가 페이지 테이블을 위해서 필요할 것이다. 이렇게 된다면 배보다 배꼽이 큰 상황이 되기 때문에 보통의 운영체제는 페이지를 바이트 단위가 아닌 그보다 큰 단위로 관리한다. 이 단위를 페이지 크기라고 부른다. 할당 단위는 이러한 페이지를 관리하는 테이블을 몇 개씩 할당하는 지를 결정하는 단위가 된다. 이들은 시스템에 따라 가변적으로 관리된다. 윈도우에서

Copyright © 2011, 신영진

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

이러한 정보를 구하기 위해서는 GetSystemInfo라는 함수를 사용하면 된다.

void WINAPI GetSystemInfo(LPSYSTEM_INFO lpSystemInfo);

GetSystemInfo의 원형은 위와 같다. lpSystemInfo의 dwPageSize가 페이지 크기, dwAllocationGranularity가 할당 단위가 된다. lpMinimumApplicationAddress와 lpMaximumApplicationAddress는 각각 응용 프로그램에서 접근할 수 있는 최소 메모리 주소와 최대 메모리 주소를 담고 있다.

가상 메모리 해제

VirtualFree, VirtualFreeEx 함수는 할당된 가상 메모리를 해제하는 작업을 한다. 앞서 살펴본 할당 함수처럼 Ex함수는 추가적으로 특정 프로세스의 가상 메모리를 해제하는 기능을 가지고 있다. VirtualFree 함수의 원형은 다음과 같다.

BOOL VirtualFree(LPVOID lpAddress, DWORD dwSize, DWORD dwFreeType);

lpAddress에는 해제하고자 하는 가상 메모리의 시작 번지를 입력해 준다.

두 번째 인자인 dwSize에는 해제하려고 하는 가상 메모리의 크기를 입력해준다. 여기서 주의해야 할 점이 있다. 만약 dwFreeType이 MEM_RELEASE인 경우에는 이 값을 0으로 지정해야 한다. 그러면 할당된 모든 메모리가 해제된다. 반대로 dwFreeType이 MEM_DECOMMIT인 경우에는 해제하고 싶은 메모리 크기를 입력하면 된다. 이 경우에는 해당 메모리를 포함하는 모든 페이지가 예약 상태로 바뀐다.

마지막 인자는 MEM_RELEASE와 MEM_DECOMMIT 중 하나를 선택해서 지정하면 된다. MEM_RELEASE를 지정하는 경우에는 VirtualAlloc시에 할당된 모든 메모리를 해제 상태로 변경하는 작업을 한다. MEM_DECOMMIT을 하면 해당 메모리를 예약 상태로 변경하는 작업을 한다.

할당과 해제

가상 메모리를 할당하고 해제하는 함수에 대해서 살펴보았다. 안타깝게도 이 두 함수는 일반적인 메모리 관리 함수보다는 복잡하기 때문에 함수 설명만 읽고는 제대로 사용하기가 쉽지 않다. 간단한 메모리 할당/해제 예를 통해서 두 함수가 동작하는 방식을 좀 더 세밀하게 살펴보도록 하자. 여기서 우리는 0x10000부터 0x30000까지는 FREE 상태의 메모리 라고 가정한다. 또한 이 시스템의 페이지 크기는 0x1000으로, 할당 단위는 0x10000이라고 가정한다.

VirtualAllocEx(process, 0x10000, 0x8000, MEM_RESERVE, PAGE_READONLY)

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

위의 호출이 이루어지고 나면 메모리는 두 개의 조각으로 나뉘게 된다. 0x10000부터 0x20000까지의 메모리는 RESERVE 상태가 되고, 0x20000부터 0x30000까지의 메모리는 FREE 상태의 메모리가 된다. 여기서 0x10000부터 0x20000까지가 예약 상태의 메모리가 되는 이유는 할당 단위가 0x10000이기 때문이다.

```
VirtualAllocEx(process, 0x12000, 0x1000, MEM_COMMIT, PAGE_READWRITE);
VirtualAllocEx(process, 0x17000, 0x128, MEM_COMMIT, PAGE_EXECUTE);
```

이제 메모리 구조는 좀 더 복잡해진다. RESERVE로 예약된 메모리 중에서 0x12000부터 0x13000까지는 실제 메모리에 맵핑되어서 읽기/쓰기가 가능한 메모리 영역으로 사용된다. 0x17000부터 0x18000까지의 영역 또한 실제 메모리에 맵핑되어서 실행 가능한 메모리로 사용이 된다. 두 번째 할당의 경우 0x128을 할당했지만, 가상 메모리는 페이지 단위로 관리 되기 때문에 페이지 크기의 배수 배로 조정되어서 0x1000 만큼이 할당된 것이다. 여기까지 이루어진 메모리 할당 구조를 그림으로 그려보면 <그림 3>과 같이 총 6개의 구역으로 나뉜다는 것을 알 수 있다.

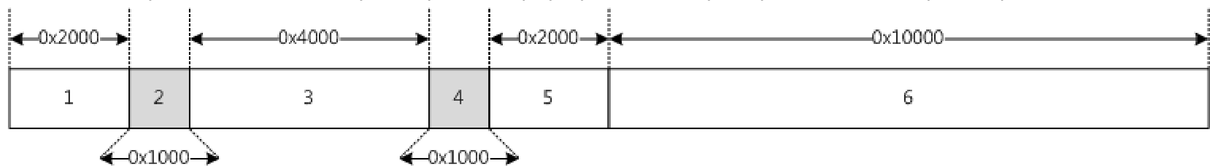

그림 3 메모리 레이아웃

```
VirtualAllocEx(process, 0x20000, 0x8000, MEM_RESERVE, PAGE_EXECUTE);
VirtualAllocEx(process, 0x20000, 0x1000, MEM_COMMIT, PAGE_READWRITE);
```

메모리 구조를 조금 복잡하게 만들기 위해서 위와 같은 두 번의 할당을 더 한 경우의 메모리 구조를 생각해보자. 이 상태의 메모리 레이아웃을 그려보면 <그림 4>와 같이 된다.

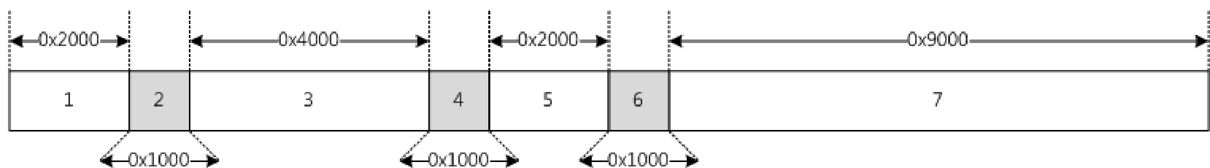

그림 4 메모리 레이아웃

이제 할당된 메모리의 해제 작업을 해 보도록 하자. 해제 작업을 하기에 앞서서 먼저 이해해야 하는 것은 각 메모리 구역의 할당 단위다. 1,2,3,4,5 구역은 구분되어 있지만 실상은 최초의 MEM_RESERVE를 통해서 확보된 하나의 할당 구역에 묶여 있다. 마찬가지로 6,7도 하나의 할당 구역에 묶여 있다. 이와 같이 동일한 할당 구역에 존재하는 메모리는 MEM_RELEASE를 사용하면 한번에 모두 해제(FREE) 상태로 변경할 수 있다.

Copyright © 2011, 신영진
이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

다음과 같이 호출하면 1,2,3,4,5 구역에 있는 메모리는 모두 한 번에 해제(FREE) 상태가 된다. 이 경우에 한 할당 단위에 포함된 메모리 중에 할당(COMMIT) 상태의 구역이 있는 것은 문제가 되지 않는다. 왜냐하면 VirtualFree 함수가 내부적으로 모두 예약(RESERVE) 상태로 변경한 다음 해제(FREE)로 변경하는 작업을 해주기 때문이다. MEM_RELEASE를 호출할 때 기억해야 할 사항은 딱 두 가지다. 하나는 lpAddress에 MEM_RESERVE를 할 때의 시작 주소를 넣어주어야 한다는 점이고, 다른 하나는 dwSize에 0을 넣어야 한다는 점이다.

```
VirtualFreeEx(process, 0x10000, 0, MEM_RELEASE);
```

세부적으로 할당(COMMIT)된 메모리를 예약(RESERVE) 상태로 변경하는 방법을 살펴보자. 6번 구역 메모리를 예약(RESERVE) 상태로 변경하기 위해서는 다음과 같이 호출하면 된다.

```
VirtualFreeEx(process, 0x20000, 0x1000, MEM_DECOMMIT);
```

이 때 기억해야 할 점은 MEM_DECOMMIT은 lpAddress부터 dwSize 크기만큼 사이에 포함된 모든 COMMIT된 메모리를 RESERVE 상태로 변경시킨다는 점이다. 즉, 다음과 같은 호출 또한 6번 구역을 RESERVE 상태로 만들 수 있다는 것이다.

```
VirtualFreeEx(process, 0x20000, 0x1, MEM_DECOMMIT);
VirtualFreeEx(process, 0x1F000, 0x1200, MEM_DECOMMIT);
```

그렇다면 다음과 같은 VirtualFreeEx 함수 호출은 위의 페이지 상태를 어떻게 변화시킬까? 이는 2, 4번 페이지를 모두 예약(RESERVE) 상태로 변경하는 것을 의미한다.

```
VirtualFreeEx(process, 0x12000, 0x5000, MEM_DECOMMIT);
```

가상 메모리 상태 조회 및 보호 속성 변경

VirtualQuery, VirtualQueryEx 함수는 특정 메모리 영역의 가상 메모리 정보를 반환해 주는 역할을 한다. 특정 메모리 영역을 우리가 사용할 수 있을지 판단할 때 사용하면 된다. VirtualQuery 함수의 원형은 다음과 같다.

```
SIZE_T VirtualQuery(LPCVOID lpAddress, PMEMORY_BASIC_INFORMATION lpBuffer, SIZE_T dwLength);
```

lpAddress에는 정보를 알고 싶은 메모리의 주소를 입력한다.

Copyright © 2011, 신영진

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

lpBuffer에는 가상 메모리 정보가 담길 버퍼를 입력해 준다. 일반적으로 MEMORY_BASIC_INFORMATION 구조체를 전달해주면 된다. MEMORY_BASIC_INFORMATION 구조체는 다음과 같은 구조로 되어 있다.

```
typedef struct _MEMORY_BASIC_INFORMATION
{
	PVOID BaseAddress;	// lpAddress가 포함된 메모리 구역의 시작 위치
	PVOID AllocationBase;	// 할당 시작 위치
	DWORD AllocationProtect;	// 할당될 때의 보호 속성
	SIZE_T RegionSize;	// 할당된 메모리의 크기
	DWORD State;	// FREE, RESERVE, COMMIT 등의 메모리 상태
	DWORD Protect;	// 현재 해당 메모리 구역의 보호 속성
	DWORD Type; // 메모리의 타입을 나타낸다(<표 4> 참고).
} MEMORY_BASIC_INFORMATION,	*PMEMORY_BASIC_INFORMATION;
```

표 4 메모리 타입

값	의미
MEM_IMAGE	실행 파일이 로딩되어 사용되는 메모리
MEM_MAPPED	메모리 맵 파일로 사용되는 메모리
MEM_PRIVATE	VirtualAlloc/VirtualAllocEx 등을 사용해 할당 받은 메모리

dwLength에는 lpBuffer의 크기를 입력해준다. 보통의 경우 sizeof(MEMORY_BASIC_INFORMATION)을 지정하면 된다.

VirtualQuery 함수의 사용법은 어렵지는 않지만 넘어오는 구조체의 정보를 정확하게 이해하기 위해서는 약간의 부연 설명이 필요할 것 같다. MEMORY_BASIC_INFORMATION 구조체의 내용 중에 AllocationBase와 BaseAddress, 그리고 AllocationProtect와 Protect의 차이점이 가장 궁금할 것이다. 앞쪽에 Allocation이 붙은 것은 메모리가 RESERVE 될 때의 단위를 말한다. 그리고 그런 것이 없는 것은 COMMIT되거나 COMMIT 됨으로 해서 분리된 공간들의 구역을 나타낸다.

앞쪽의 메모리 할당 예제를 통해서 살펴보는 것이 좀 더 쉽게 이해하는 방법이다. <그림 4>와 같이 할당된 메모리를 생각해보자. 우선 2번 구역에 속한 임의의 포인터에 대해서 VirutalQueryEx를 수행하면 그 결과가 어떻게 나올까? AllocationBase는 0x10000, Base는 0x12000, AllocationProtect는 PAGE_READONLY, Protect는 PAGE_READWRITE가 나온다. 그리고 끝으로 RegionSize는 0x1000이 된다. 1, 2, 3, 4, 5 구역에 속한 임의의 포인터를 질의하더라도 모두 AllocationBase는 0x10000, AllocationProtect는 PAGE_READONLY가 나온다.

Copyright © 2011, 신영진

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

끝으로 가상 함수의 보호 속성을 변경하는 VirtualProtect 함수에 대해서 살펴보자. 이 함수는 설정된 페이지의 보호 속성을 변경하는 역할을 한다. 원형은 아래와 같다.

BOOL VirtualProtect(LPVOID lpAddress, SIZE_T dwSize, DWORD flNewProtect, PDWORD lpflOldProtect);

lpAddress에는 변경할 페이지의 시작 주소를, dwSize에는 페이지 크기를 입력하면 된다. fNewProtect에는 변경할 보호 속성을 지정하고, lpflOldProtect는 현재 보호 속성을 리턴 받기 위해 사용된다. 현재 보호 속성을 리턴 받지 않아도 되는 경우라도 반드시 합당한 포인터를 전달해야 한다.

vmwalk

<리스트 1>에는 특정 프로세스의 가상 메모리 맵을 보여주는 vmwalk 프로그램의 소스가 나와 있다. vmwalk <pid>와 같은 형태로 사용하면 pid에 해당하는 프로세스의 메모리 영역을 보여준다. 앞서 설명한 함수들을 실제로 어떻게 사용하는지에 초점을 맞춰서 보도록 하자.

리스트 1 vmwalk 소스

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

LPVOID GetPtr(LPVOID address, SIZE_T offset)
{
    return (LPVOID)((DWORD_PTR) address + offset);
}

int _tmain(int argc, TCHAR *argv[])
{
    DWORD pid;

    if(argc < 2)
        pid = GetCurrentProcessId();
    else
        pid = _tcstoul(argv[1], NULL, 10);

    HANDLE process = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, pid);
    if(!process)
    {
        printf("프로세스 열기 실패\n");
        return 0;
    }

    printf("가상 메모리 레이아웃: %d\n\n", pid);
```


이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

```
    SYSTEM_INFO si;
    GetSystemInfo(&si);

    LPVOID low = si.lpMinimumApplicationAddress;
    LPVOID high = si.lpMaximumApplicationAddress;

    printf("접근 가능 메모리   : 0x%08X ~ 0x%08X\n", low, high);
    printf("메모리 할당 단위   : 0x%08X\n", si.dwAllocationGranularity);
    printf("메모리 페이지 크기 : 0x%08X\n\n", si.dwPageSize);


    char *protect;
    char *state;
    BOOL guard, nocache, writeCombine;
    MEMORY_BASIC_INFORMATION mbi;

    LPVOID addr = low;
    while(addr < high)
    {
        if(VirtualQueryEx(process, addr, &mbi, sizeof(mbi)) != sizeof(mbi))
        {
            printf("VM 질의 실패\n");
            break;
        }

        switch(mbi.State)
        {
        case MEM_FREE: state = "FREE"; break;
        case MEM_RESERVE: state = "RESERVE"; break;
        case MEM_COMMIT: state = "COMMIT"; break;
        default: state = "?"; break;
        }

        guard = mbi.Protect & PAGE_GUARD;
        nocache = mbi.Protect & PAGE_NOCACHE;
        writeCombine = mbi.Protect & PAGE_WRITECOMBINE;
        mbi.Protect &= ~(PAGE_GUARD | PAGE_NOCACHE | PAGE_WRITECOMBINE);

        switch(mbi.Protect)
        {
        case PAGE_READONLY: protect = "Read"; break;
        case PAGE_READWRITE: protect = "Read/Write"; break;
        case PAGE_EXECUTE: protect = "Execute"; break;
        case PAGE_EXECUTE_READ: protect = "Read/Execute"; break;
        case PAGE_EXECUTE_READWRITE: protect = "Read/Write/Execute"; break;
        case PAGE_NOACCESS: protect = "No Access"; break;
        case PAGE_WRITECOPY: protect = "Copy On Write"; break;
        case PAGE_EXECUTE_WRITECOPY: protect = "Execute/Copy On Write"; break;
        default: protect = "?"; break;
        }
```

Copyright © 2011, 신영진

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/

```
        printf("0x%08X ~ 0x%08X %8s %22s %s %s %s₩n"
                , mbi.BaseAddress
                , (DWORD_PTR)mbi.BaseAddress + mbi.RegionSize
                , state
                , protect
                , guard ? "Guard" : ""
                , nocache ? "No Cache" : ""
                , writeCombine ? "Write Combine" : "");

        addr = GetPtr(addr, mbi.RegionSize);
    }

    CloseHandle(process);
}
```

참고자료

찰스 페졸드의 Programming Windows, 5th Edition
Charles Petzold, 한빛미디어

Windows API 정복
김상형, 가남사

Copyright © 2011, 신영진

이 문서는 Creative Commons 라이센스를 따릅니다.
http://creativecommons.org/licenses/by-nc-nd/2.0/kr/