요즈음의 컴퓨터는 대략 512M정도의 물리메모리를 사용한다. 하지만 이 메모리양은 많은 프로그램을 동시에 실행하기 에는 그리 많은 메모리가 아니다. 그렇다고 값이 비싼 물리 메모리를 무조건 많이 사용할 수 도 없다.그래서 대부분의 운영체제는 하드 디스크의 특정 공간(PageFile.sys)을 마치 물리 메모리 인것처럼 사용한다. 이를 가상메모리라고 한다.물리메모리가 부족할 때 물리 메모리의 하드디스크로 잠시 저장하고 확보된 물리메로리를 계속 사용할 수 있게되는 것이다.(이를 Swapping이라고 한다.)

제어판에서 시스템을 선택후 고급탭에서 성능항목의 설정 버튼을 누른후 나타나는 윈도우에서 고급탭에 있는 가상메모리 변경을 선택하면 위의  그림 처럼 가상 메모리를 변경할 수 있다

하나의 프로그램이 실행될 때 운영체제는 해당하는 프로그램이 사용할 수 있는 4G의 가상 주소 공간을 만든다.아래 그림에서 프로그램 A가 실행 될 때 A는 4G의 자신만의 주소 공간을 갖게 된다. 이 주소는 Page Table과 Page directory 라는 것을 사용해서 실제 물리 주소로 변경된다.

위의 그림에서,

결국 A와 B라는 2개의 프로세스가 실행중인때 A프로세스의 1234'0000이라는 주소는 B프로세스의 1234'000주소와는 전혀 다른 내용을 가르키게 되는 것이다.

그러므로 결국 A가 아무리 잘못된 포인터 연산을 하더라도 B프로세스의 메모리 영역은 안전할 수 있는 것이다.

하나의 프로세스가 사용하는 4G의 주소공간중 상위 2G(0x8000'0000~0xFFFF'FFFF)는 System에의해 사용되어 진다. 주로 이영역에 운영체제 Code와 각종 Device Driver Code가 놓여진다. 또한 이공간은 모든 프로세스들이 공유하는 공간이다.하지만 하위 2G(0x0000'0000 ~ 0x7FFF'FFFF)는 해당 프로세스가 독립적으로 사용할 수 있는 영역이며 모든 프로세스가 독립적으로 사용하는 Private공간이다.

**GlobalMemoryStatus()**라는 함수를 사용하면 현재 물리메모리의 크기와 Paging파일의 크기, 가상주소 공간의 크기등을 구할 수 있다.

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

#define DIV (1024\*1024)

const char\* division = "MB";

#define dprint(expr) printf(#expr " = %d %s\n", (expr/DIV)+1, division)

void main()  
{  
    MEMORYSTATUS ms;  
    ms.dwLength = sizeof( ms);

    GlobalMemoryStatus( &ms );

    dprint( ms.dwTotalPhys );  
    dprint( ms.dwAvailPhys );  
    dprint( ms.dwTotalPageFile);  
    dprint( ms.dwAvailPageFile);  
    dprint( ms.dwTotalVirtual);  
    dprint( ms.dwAvailVirtual);  
}

2개의 프로세스가 동일한 DLL을 사용한다고 가정해 보자.

위의 그림 1처럼 2개의 프로세스는 물리메모리 공간을 공유 하고 있게 될것이다.

또한 x라는 변수가 PAGE B에 있다고 가정하가.

이때 B 프로세스가 변수 x의 값을 변경했다고 하면 A 프로세스에서 B 가 변경한 값을 읽을 수 있을까 ?

(즉, x 는 A와 B 사이에서 공유 될까 ?)

답은 틀린다. DLL 의 내의 각 PAGE는 PAGE\_WRITECOPY 보호속성을 지니고 있다. 그래서 B 프로세스가 쓰려고 하는 순간 물리 메모리의 또 다른 공간에 해당 Page가 복사(PAGE C)되고 복사된 Page가 쓰려고 한 프로세스의 가상주소와 연결된다.(그림 2)

그러므로 Process B와 프로세스 A가 가진 PAGE 2는 공유되지 않는다.

이런 방법을 사용하므 로서 10개의 메모장이 동시에 실행된다 하더라도 10개의 instance는 동일한 물리메모리를 공유하게 된다. 그러다가 어떤 instance가 특정 Page를 쓰려고 하는 순간 해당 Page의 사본이 생성 되어 쓰기를 시도한 instance에 연결되는 것이다.

이런 기법은 “**lazy evaluation**” 이라고도 불리우는데, 실제 필요할 때 까지 물리메모리를 절약할 수 있게 해준다.

마지막으로 VirtualAlloc 또는 VirtualAllocEx 함수로 메모리를 할당할 때는 PAGE\_WRITECOPY 혹은 PAGE\_EXECUTEWRITECOPY 속성을 사용하면 안 된다. (실패 할것이다. GetLastError :ERROR\_INVALID\_PARAMETER)

이 두 속성은 운영체제가 exe혹은 dll 파일을 맵 할 때 사용한다.

.exe 혹은 DLL 을 만들때, 컴파일러가 링커가 만들어 내는 표준 섹션 이외에 사용자가 임의로 섹션을 추가할 수 있다.

아래 지시어는 “.Shared” 라는 이름의 섹션을 만들어 내는 코드이다.

#pragma data\_seg(“.shared”)

long c = 0; // must initialized

#pragma data\_seg()

|  |
| --- |
| 다음의 함수를 사용해서 프로세스의 가상 주소 공간을 할당할 수 있다. |
| **LPVOID** **VirtualAlloc(** **LPVOID** *lpAddress***, SIZE\_T** *dwSize***, DWORD** *flAllocationType***, DWORD** *flProtect* **);** |
| 가상 주소 공간에 메모리를 할당 할 때는 아래의 2가지 과정이 필요하다. |
| Reserve : 물리 공간을 확보하지 않고 단지 가상 주소 공간만을 확보한다. 아직 사용할 수는 없다.  Commit : 예약된 가상주소 공간에 물리 공간을 연결한다. |
| 예약과 확정을 동시에 할 수도 있다. |
| 다음은 VirtualAlloc의 인자에 대한 설명이다. |
| [in] *lpAddress* |
| 할당하고자 하는 메모리의 시작 주소를 지정한다. 메모리를 예약하는 경우 반드시 64KB의 배수로 지정해야 한다. 0을 지정하는 경우 시스템이 임의로 지정한다. |
| [in] *dwSize* |
| 할당하고자 하는 메모리의 크기를 지정한다. |
| [in] *flAllocationType* |
| [in] *lProtect* |
| 할당할 메모리의 보호 속성을 지정한다. |

|  |
| --- |
| **Memory Mapped File** |
| **MMF 개념** |
| 하나의 프로세스는 실행 중에 4G의 가상 주소 공간을 사용한다. 가상주소의 공간의 한 페이지는 아래 그림 처럼 실제 물리 메모리에 매핑 되거나 가상메모리(Paging file)에 매핑 된다.  이때, 가상 주소의 한 공간이 물리메모리나 페이징 파일이 아닌 **일반 파일에 매핑** 되도록 할 수도 있다. 이름 메모리 맵 파일이라고 한다. 메모리 맵 파일을 사용하면 파일의 접근을 마치 메모리에서 읽고 쓰는 것처럼 할 수 있다. |
|  |
| MMF를 사용 하는 이유는 주로 아래와 같다. |
| 1. 파일 작업을 보다 쉽고 빠르게 할 수 있다.( R/W 작업시 버퍼가 필요 없다.) 2. 2개 이상의 프로세스가 메모리를 공유 해서 IPC기법으로 활용된다. |
|  |
| Memory Mapped File 만들기 |
| MMF 를 사용하려면 아래의 단계를 거쳐야 한다. |
| 1. 메모리에 맵 하고자 하는 파일을 Open(또는 Create) 한다. 2. 파일 매핑 커널 Object를 생성한다. 3. 매핑 커널 오브젝트를 프로세스의 가상주소에 연결한다. 4. 가상주소에 연결된 Memory Mapped File 을 사용한다. 5. 가상주소 공간을 해지한다. 6. 파일 매핑 커널 오브젝트를 닫는다. 7. 파일을 닫는다. |