

Chapter-2

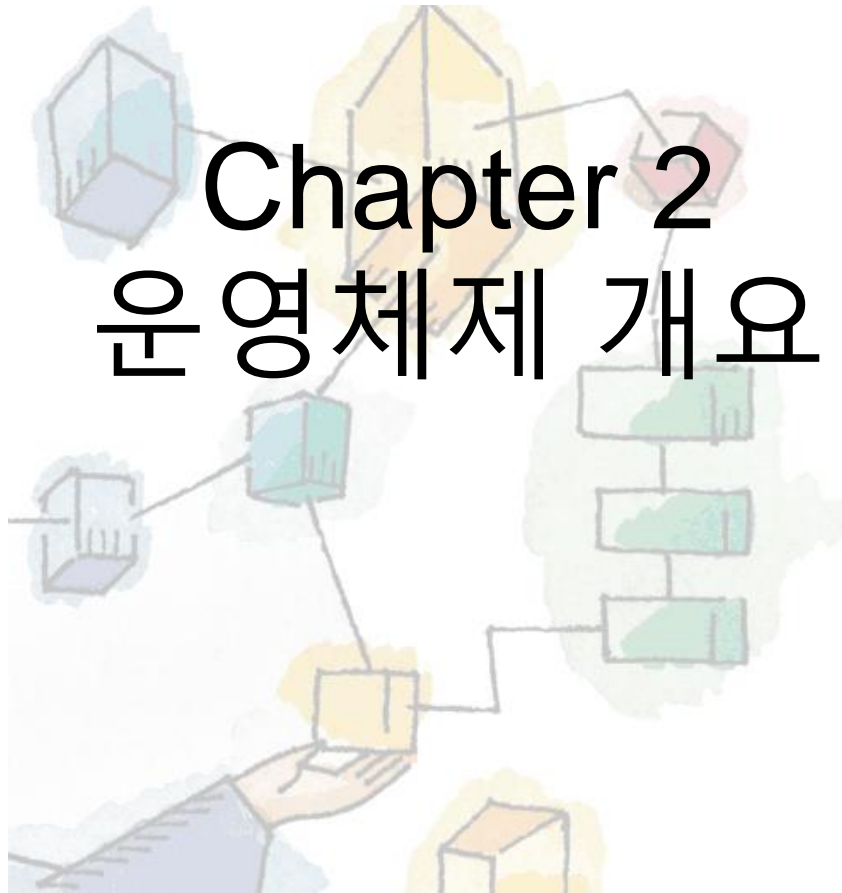
운영 체제

정내훈

2023년 가을학기
게임공학과
한국공학대학교

Chapter 2

운영체제 개요



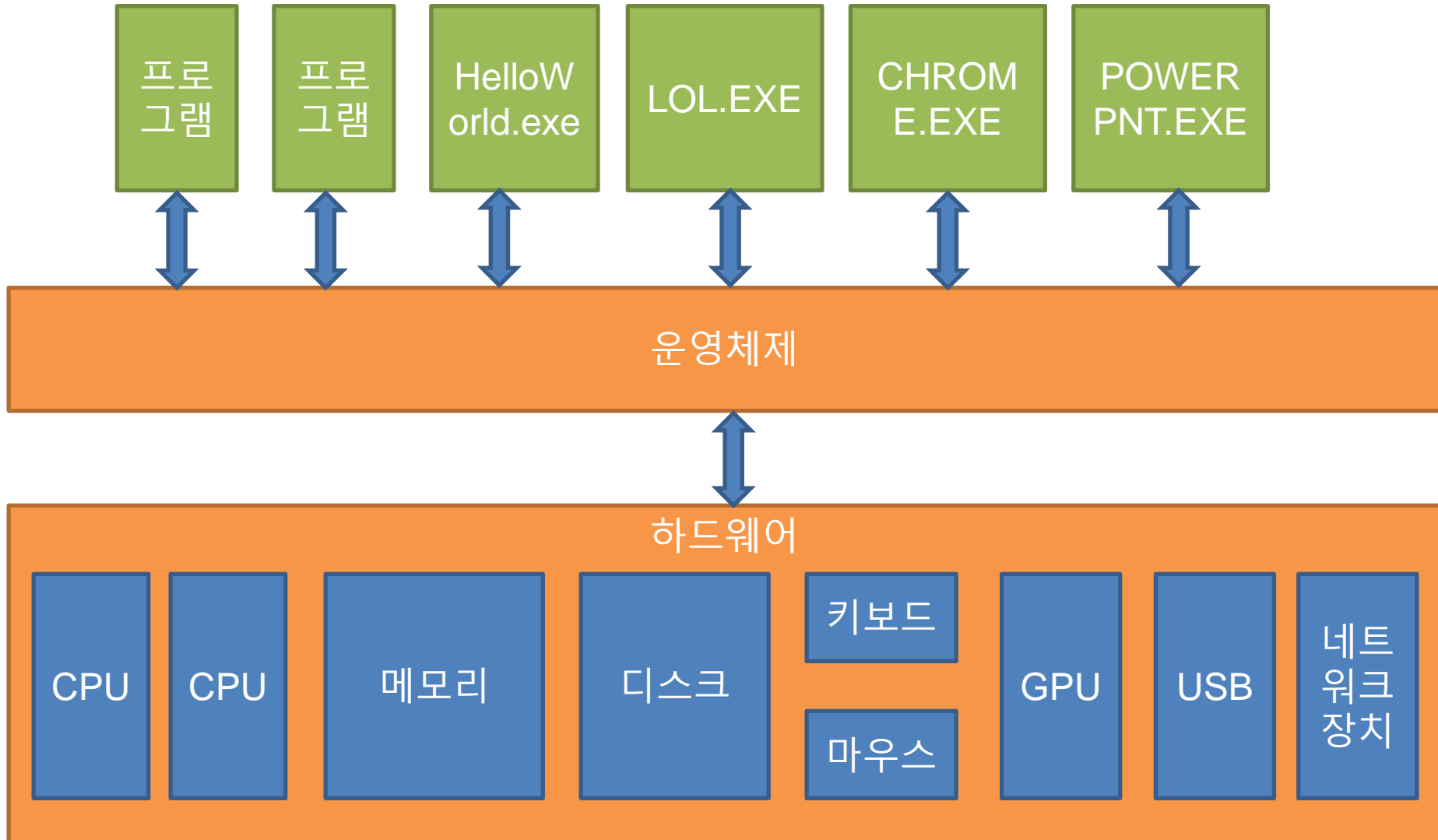


소개

- 운영체제가 하는 일 : 컴퓨터를 쉽게 사용할 수 있게 해주기
 - 프로그램을 쉽게 실행 (HW 관리)
 - 프로그램간의 메모리 공동 사용 (자원공유/분배)
 - 장치와 상호작용 기능 제공 (입출력)
- 위와 같은 일을 하는 소프트웨어를 운영체제라 부른다.
 - 프로그램과 하드웨어사이에 존재
 - 프로그램이 하드웨어를 사용할 때 반드시 운영체제를 통하도록 강제한다. (CPU 설계 시)



소개





가상화(Virtualization)

- 실제(Physical) 자원을 사용하기 쉽도록 가공해서 제공하는 것
 - 실제 자원 : CPU, 메모리, 디스크
 - i3, i5, i7, 콰두, XEON, EPYC 등 수많은 종류의 CPU가 있으나 우리는 그것을 고려하지 않음.
 - 메모리 크기를 고려해서 프로그래밍을 하지 않음
 - 나중에는 하게 됨
 - 파일을 읽을 때 : HDD, SSD, CD, DVD, USB메모리, 네트워크 드라이브를 구분하지 않음.
 - (운영체제를 가상머신이라고 부르기도 함?? : 그런 사람 별로 없음)



시스템 콜(System Call)

- CPU, 디스크를 날 것으로 컨트롤 하는 것은 너무 어려움, 쉽게 사용 할 수 있도록 제공하는 간단한 표준 API

```
char word[100];  
FILE *fp = fopen("readme.txt", "rw");  
fscanf(fp, "%s", word);
```

V.S.

컴퓨터 메모리에서 사용하지 않는 100Byte를 탐색
다른 프로그램이 사용하지 못하도록 사용 중 표시
디스크의 목록 부분을 읽는다. (메모리 할당 필요)

- Disk의 전원을 넣는다.
- 모터를 회전 시킨다. -> 정속도에 도달할 때까지 기다린다.
- 헤드를 목록의 위치로 이동 시킨다. -> 도달 할 때까지 기다린다.
- DMA에 전송 명령을 내린다.
- 헤드에 읽기 명령을 내린다.
- DMA 전송이 종료할 때 까지 기다린다.

목록에서 readme.txt 객체 검색

객체에서 본문의 주소 추출

디스크의 해당 주소를 읽는다. (메모리 할당 필요)

앞 부분에서 문자열을 하나 복사한다.



자원 관리자 (Resource Manager)

- 여러 개의 프로그램을 동시에 실행할 때
자원의 효율적인 배분과 공유, 재사용을
담당
 - 이를 위해 CPU에서 사용자 프로그램의 직접
적인 자원 접근을 막고 있음.
 - CPU, 메모리, 디스크, 프린터, GPU, 네트워크



CPU 가상화

- 작업 관리자를 실행해 보자
 - 많은 프로그램이 실행되고 있다.
 - 프로그램을 작성하면서 다른 프로그램에게 CPU를 양보 한 적이 있는가?
- 마치 컴퓨터에 수십 개의 CPU가 존재한다는 가상화가 있다.
 - CPU를 무한한 개수의 CPU로 증식
 - 이를 통해 많은 프로그램을 즉시 실행 시킴

CPU 가상화

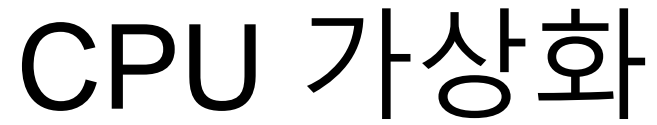


CPU 가상화

- 다음 프로그램을 실행해 보자

```
#include <iostream>
using namespace std;

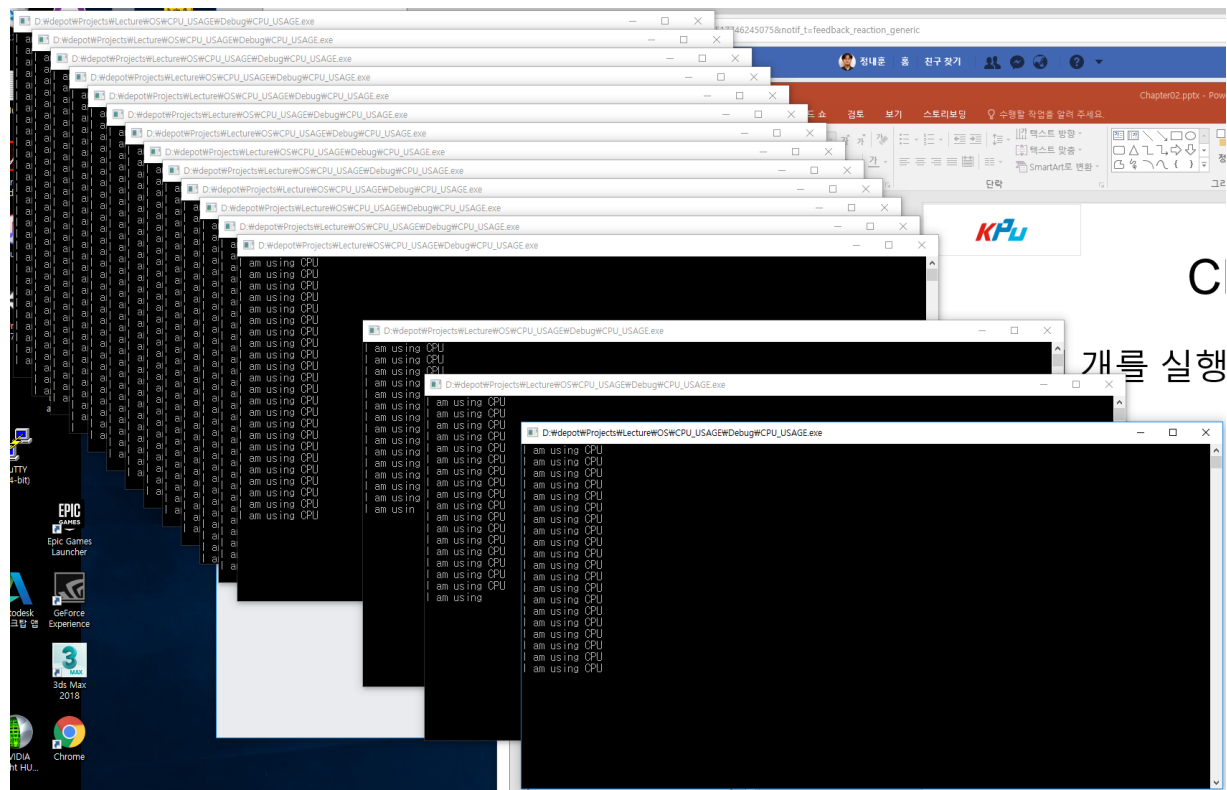
int main()
{
    while (true) {
        volatile int sum = 0;
        for (auto i = 0; i < 2000000000; ++i)
            sum = sum + i;
        cout << "I am using CPU\n";
    }
}
```

[illegible]



CPU 가상화

- 여러 개를 실행해 보자

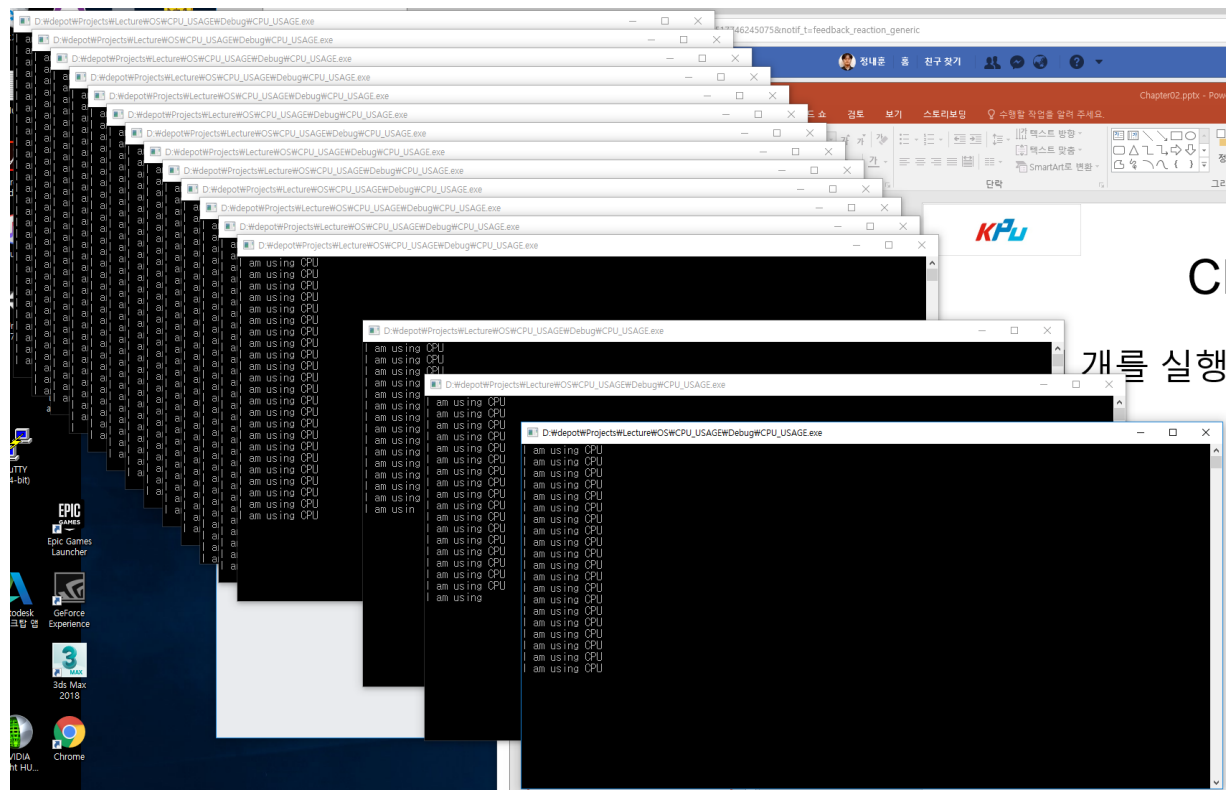


CPU는 **하나** 지만 여러 개의 프로그램이 **동시에** 실행된다.



CPU 가상화

- 여러 개를 실행해 보자

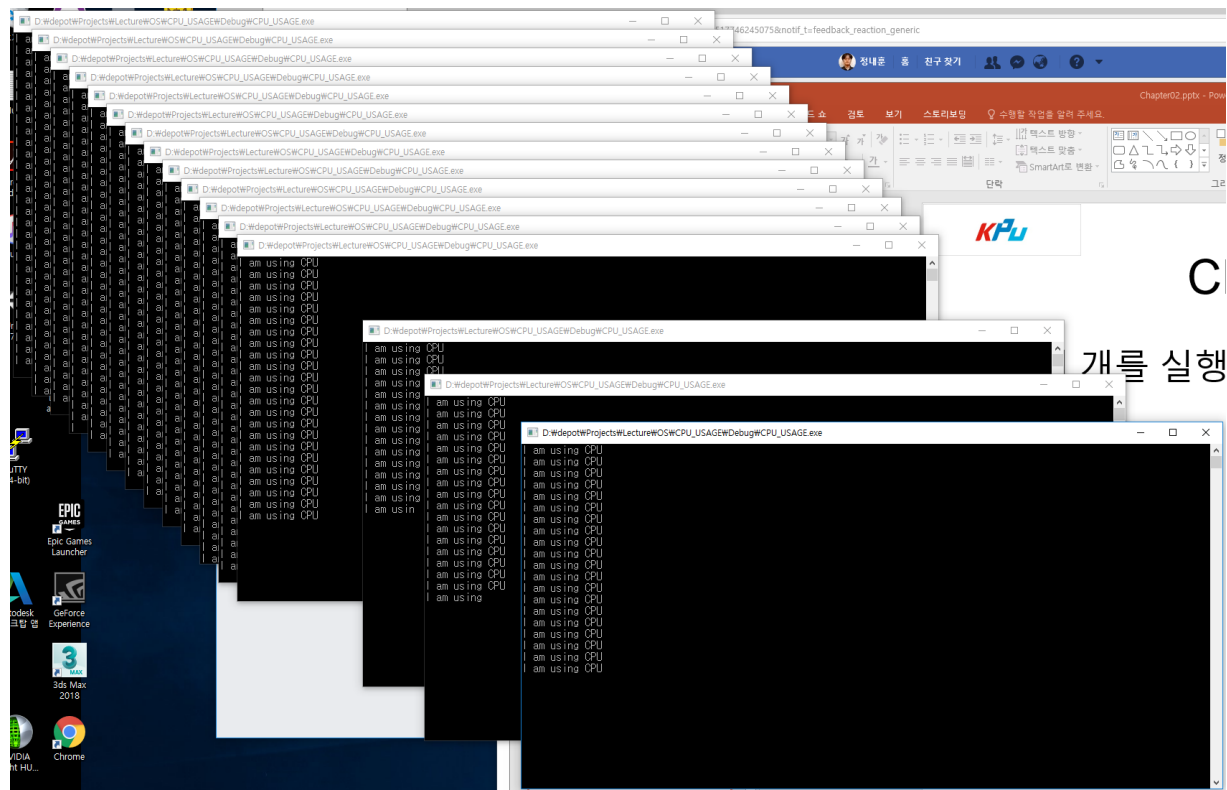


코어는 **네개**지만 16개의 프로그램이 **동시**에 실행된다.



CPU 가상화

- 여러 개를 실행해 보자

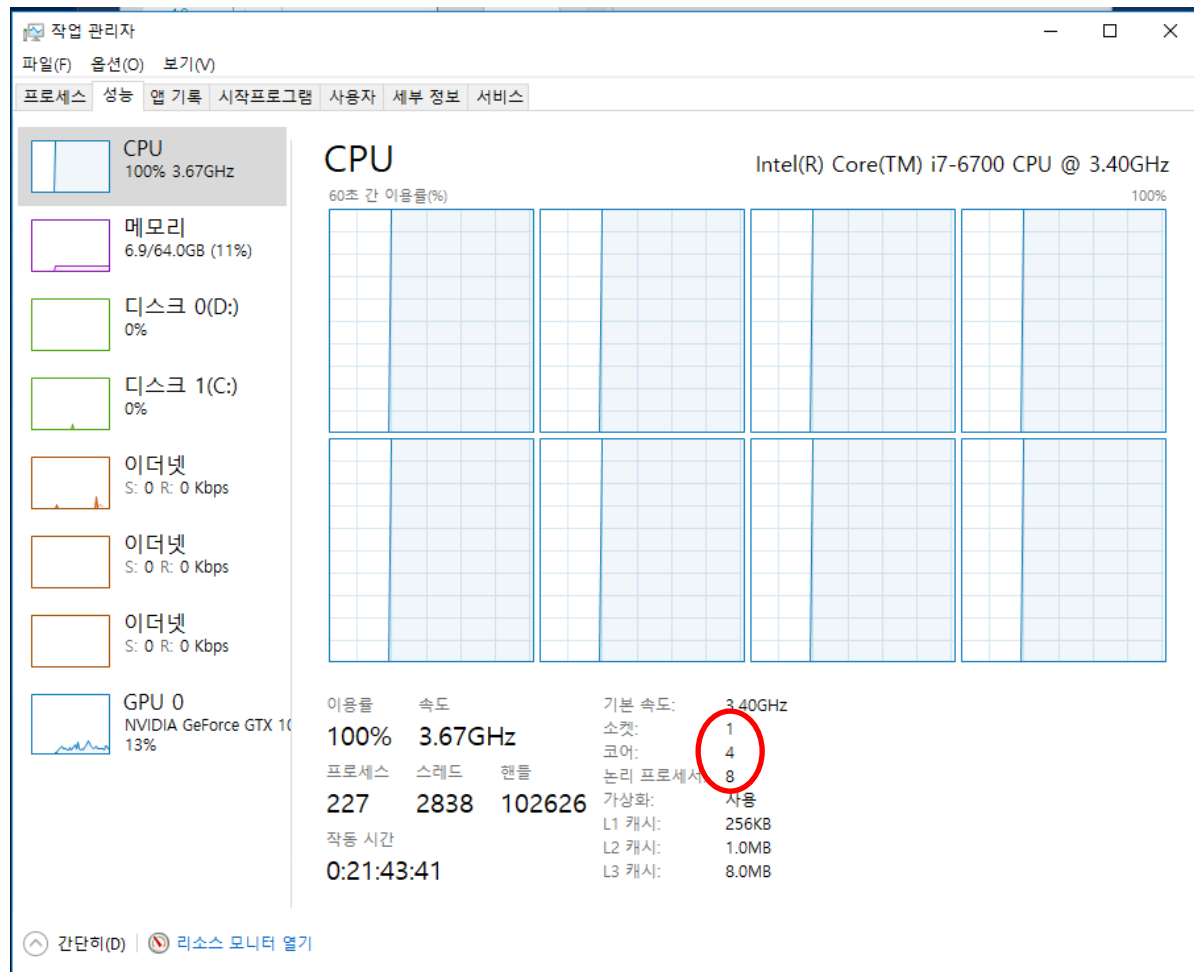


논리코어는 8개지만 16개의 프로그램이 동시에 실행된다.



CPU 가상화

- 컴퓨터의 상태





메모리 가상화

- 실제 메모리는 Byte의 배열
 - 배열 : C 언어의 그것, array
- 프로그램의 모든 자료 구조는 메모리에 있다. (Disk에 있는 데이터도 일단 메모리에 읽어 놔야 읽을 수 있다.)
 - 메모리 읽기 (Read, Load):
 - 주소를 주고 데이터를 얻기
 - 메모리 쓰기 (Write, Store):
 - 주소와 데이터를 주고 쓰게 하기.



메모리 가상화

- 문제 : 프로그램이 여러 개가 실행될 때 서로의 간섭을 어떻게 피할까?
- 답 : 가상화
 - 프로그램은 모든 메모리를 혼자 전부 사용한다는 착각속에 실행됨.
 - 다른 프로그램의 메모리를 아예 볼 수도 없음



메모리 가상화

- 테스트 프로그램

```
#include <iostream>
using namespace std;

int main()
{
    int *p = new int; // int 한개를 얻음
    // 주소 출력
    cout << "Address of P: " << p << endl;
    *p = 0; // 초기화
    while (true) {
        int sum = 0;
        for (auto i = 0; i < 4000000000; ++i) sum = sum + i;
        *p = *p + 1;
        cout << dec << " p: " << *p << endl; // p의 값을 출력
    }
}
```



메모리 가상화

- 결과

```
D:\depot\Projects\Lecture\OS\MEMORY_USAGE\x64\Debug\MEMORY_USAGE.exe
Address of P: 4e1650
p: 1
p: 2
p: 3
p: 4
p: 5
p: 6
p: 7
```



메모리 가상화

- 여러 개를 수행 했을 때 결과

```
D:\depot\Projects\Lecture\OS\MEMORY
Address of P: 57fc20
p: 1
p: 2
p: 3
```

```
D:\depot\Projects\Lecture\OS\MEMORY
Address of P: 57fc20
p: 1
p: 2
p: 3
```

조작

```
D:\depot\Projects\Lecture\OS\MEMORY
Address of P: 57fc20
p: 1
p: 2
p: 3
```

```
D:\depot\Projects\Lecture\OS\MEMORY
Address of P: 57fc20
p: 1
p: 2
p: 3
```



메모리 가상화

- 여러 개를 수행 했을 때 실제 결과

The screenshot shows a debugger window with multiple instances of a program running. Each instance has a 'MEMORY_USAGE.exe' window open, displaying a list of memory addresses (p: 1 to p: 15) and their corresponding values. The addresses are: 5c13c0, 5a5a20, 531390, and 416740. The values are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15. The debugger window title is 'D:\depot\Projects\Lecture\OS\MEMORY_USAGE\x64\Debug\MEMORY_USAGE.exe'.

보안 때문

WINDOW가
너무 발전했음



메모리 가상화

- **조작**을 믿으세요
 - 같은 주소에 +1을 하는 프로그램이 여러 개인데 왜? 1 씩 증가하는가?
- 모든 실행중인 프로그램은 각자 고유의 메모리를 갖고 있다.
 - 같은 주소를 읽고 쓰지만 프로그램이 다르면 다른 메모리이다.
 - 모든 프로그램은 같은 주소에서 실행된다.
 - 서로 주소가 겹친다.



메모리 가상화

- 전문용어를 사용하면
- 각 프로세스는 각자의 가상 주소 공간을 갖는다. [아래 내용은 뒤에 자세히 설명]
 - 운영체제가 가상 주소를 실제 주소로 매핑한다. (mapping)
 - 실행중인 프로그램이 메모리를 건드릴 때 다른 프로세스의 메모리에는 영향이 없다.
 - 실제(physical) 메모리는 운영체제의 관리를 받는 공유 자원(shared resource)이다.



추가

- 메모리 가상화를 진짜로 확인하고 싶다면
 - 프로그램들이 각자 메모리를 크게 할당 받은 후
 - 겹치는 주소를 찾아서
 - 메모리 업데이트를 관찰해야 한다.
 - 하지만... 64비트 주소공간... 아마 안될꺼야...



추가

- 32비트 모드로 다음 프로그램을 실행.

```
#include <iostream>
#include <windows.h>
using namespace std;

constexpr size_t BUFF_SIZE = 1024 * 1024 * 1024 / sizeof(int);
constexpr unsigned int ADDR = 0x40000000;

int main()
{
    int* buff = new int [BUFF_SIZE];
    cout << "Address of buff : from : " << buff << " To : " << buff + BUFF_SIZE << endl;
    int* p = reinterpret_cast<int*>(ADDR);
    cout << "ADDR = " << p << endl;
    *p = 0;
    while (true) {
        Sleep(1000);
        *p = *p + 1;
        cout << " [ADDR]: " << *p << endl; // p의 값을 출력
    }
}
```