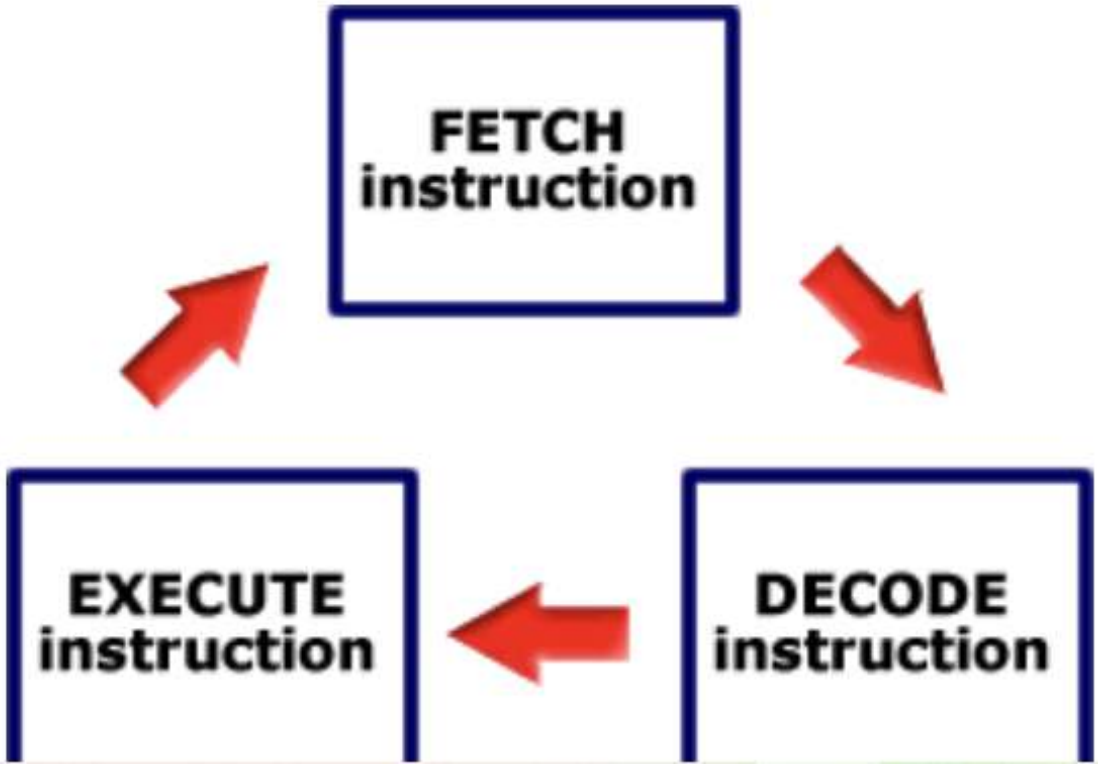
Q. What a happens when a program runs?

A running program executes instructions.

일단 실행중인 프로그램들은 명령어를 실행하는데,
기본적인 반복구조는 아래 사진과 같다.



1. fetches

-> The processor fetches an instruction from memory.
프로세서는 메모리로부터 명령어를 패치한다.

2. Decode

-> Figure out which instruction this is
어떤 명령인지 알아내기

3. Execute

-> ex) add two numbers, access memory, check a condition, jump to function ...
다양한 연산들이 여기에 해당된다. 예를 들면, 두개의 숫자를 더하는 것 & 메모리에 엑세스하는 것 등등..

4. next instruction

-> The processor moves on to the next instruction and so on.

프로세서는 다음 명령어로 이동한다.

위의 사진에서는 next instruction의 과정이 담겨져있지 않지만
가장 핵심적인 부분은 1~3번 부분이다.

Q. What is a Operating System(OS)?

Maybe..

1. Making it easy to run programs

-> 운영체제는 프로그램 실행을 용이하기 위해서 필요하다.**(자동화)**

2. Allowing programs to share memory

-> 프로그램들이 메모리를 공유할 수 있도록 허용해준다.**(제한된 메모리)**

3. Enabling programs to interact with devices

-> 하드웨어나 SSD, CPU, NIC, 메모리 등등의 장치들을
프로그램들과 상호작용할 수 있도록 중재하는 역할을 한다.

Thus, OS is in charge of making sure the system operates correctly and efficiently.
따라서, OS는 시스템이 정확하고 효율적으로 작동하는지 확인하는 역할을 담당한다.

Virtualization(가상화)

-> The OS takes a physical resource and transforms it into a virtual form of itself.
쉽게 말하면, OS는 여러 프로그램들을 컨트롤하고, 마치 개개인의 프로그램들이 물리적인 자원을 독점하는 것처럼 느끼게 해주는 것을 의미한다. 이를 '가상화'를 통해서 구현할 수 있다.

여기서 물리적인 자원은 프로세서, 메모리, 디스크 등을 의미한다.

System call

-> System call allows user to tell the OS what to do.
시스템 호출을 통해 사용자는 OS에게 무엇을 해야 하는지 알릴 수 있다.

1. Run programs (프로그램 실행)

2. Access Memory (메모리에 있는 값 읽거나 쓰기)

3. Access devices

그럼 왜? System call 이 필요한가?

-> OS는 **Kernel Space와 User Space로 구분**한다.

쉽게 말하면, **User Space**에는 우리가 흔히 아는 **Application(Zoom, PC카톡 등)**이 있고,
Kernel Space에서는 **OS가 어떻게 동작하는지에 대한 것**을 전반적으로 다룬다.

User Space에서는 Kernel Space에 접근할 수가 없는데,
이는 User Space에서 이루어지는 명령어가 System call 을 통해 OS에게 알려주기 때문이다.

The OS is a resource manager

-> OS는 CPU, 메모리와 디스크 같은 리소스를 관리하는 역할을 한다.

1. OS는 CPU를 공유해서 사용하도록 하여 여러 프로그램들이 동시에 실행할 수 있도록 해준다.

2. OS는 메모리를 공유해서 사용하도록 하여 많은 프로그램들이 명령과 데이터에 동시에 액세스 할 수 있도록 해준다.

3. OS는 디스크를 공유하도록 하여 많은 프로그램들이 디바이스들에 접근할 수 있도록 해준다.
(ex. 파일을 다운로드하고 있는 와중에 PC카톡으로 사진을 받는경우)

VIrtualizing the CPU

-> The system has a very large number of virtual CPUs.
시스템은 매우 많은 가상 CPU들을 가지고 있는데,

실제로 CPU는 1개 밖에 없지만, OS가 프로그램들한테 무한개의 CPU가 있는 것 처럼 보이게 해준다는 점이 'Virtualizing the CPU'의 핵심 포인트이다.

VIrtualizing Memory

-> 일반적으로 물리적인 메모리는 바이트의 배열로 이루어져 있다.
또한, 프로그램은 모든 데이터 구조를 메모리에 보관한다.

어떻게 메모리에 접근할 수 있을까?

1. Read memory (load)

2. Write memory (store)

그럼 delete는?
-> Delete는 Write의 일종이다.

Each process accesses its own private virtual address space.

-> 각 프로세스는 고유한 개인 가상 주소 공간에 액세스한다.
또한, OS는 주소 공간을 물리적 메모리에 매핑하는데, 여기서 물리적 메모리는 OS에서 관리하는 공유 리소스이다.

But...

The problem of Concurrency (동시성 문제)

The OS is **juggling** many things at once
한번에 여러개를 처리하는 '저글링' 현상 때문에

Modern multi-threaded programs also exhibit the concurrency problem

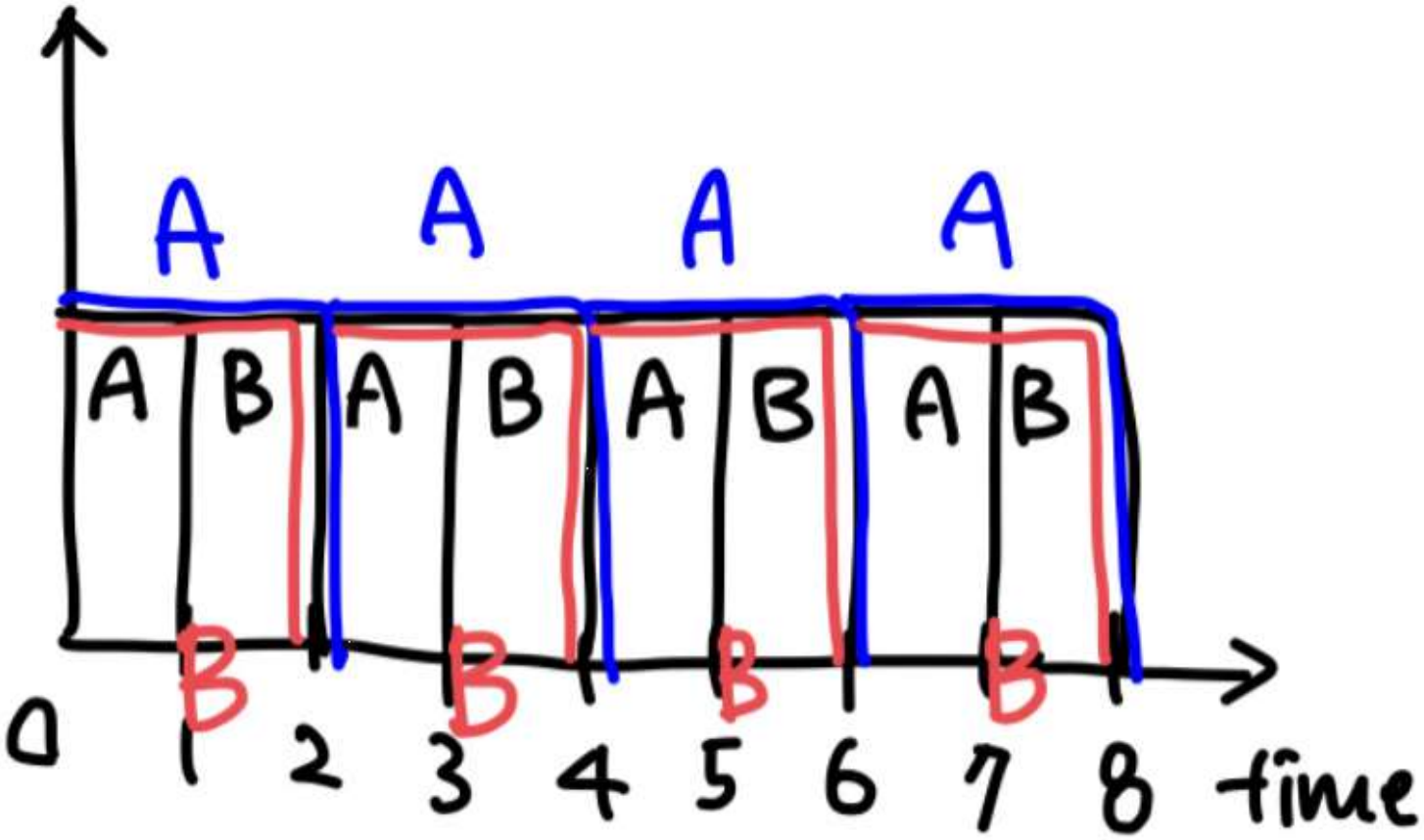
하나의 프로그램이 여러개의 CPU코어를 사용하는 멀티 쓰레드 프로그램들도 동시성 문제가 존재한다.

CPU virtualizing

-> The OS can promote the illusion that many virtual CPUs exist

어떻게 제공하는가??

바로 "**Time Sharing" (시간분할)**으로!



시간을 쪼개서 번갈아가면서 실행시켜주는 것을 의미한다.

예를 들면, A와B는 자신들이 CPU를 독점하고 있다고 생각(환상)하지만
실제로는 아니다.

Q. Process 란?

-> ***** A process is a running program.*****

프로세스는 실행되고 있는(실행 중인) 프로그램이다.

일반 프로그램은 프로세스가 아니다. 프로그램을 실행시키는 순간 프로세스가 되는 것이다.

〈프로세스의 구성요소 2가지〉

1. Memory (address space)

- instuctions

- Data section

2. Registers

-Program counter

- Stack pointer

Q. Process는 언제 생성되는가?

1st. Load a program code into memory, into the address space of the process

프로그램 코드를 메모리, 프로세스의 주소 공간에 로드한다.

- Programs initially reside on disk in executable format(**실행가능한 형태로 디스크에 존재**)

- OS perform the loading process lazily (**필요할 때만 데이터를 불러서 사용**)

2nd. The program's run-time stack is allocated

- stack은 정적 (메모리에 어떤 값이 올라갈지 정해져있는)

-

3rd. The program's heap is created (힙 생성)

- heap 은 dynamic하게 데이터를 할당하는 것(ex. malloc(), free())

4th. The OS do some other initialization tasks. (초기화 작업)

- input/output(I/O) setup

5th. Start the program running at the entry point, namely main()
(main에서 실행중인 프로그램을 시작)

Q. Process States 란?

-> A process can be one of three states

프로세스의 상태는 3가지로 구분할 수 있다

1. Runnig (실행중)

- A process is running on a processor.

여기서 짚고 넘어가야 하는 부분이 있는데,

프로그램이 실행중인것과 프로그램의 프로세스가 실행중인 것은
엄연히 다른 의미이다.

2. Ready (준비)

- A process is ready to run but for some reason the OS has chosen not to run it at this given moment

3. Blocked (막힘)

- When a process initiates an I/O request to a disk, it becomes blocked and thus some other process can use the processor

프로세스가 디스크에 대한 I/O 요청을 시작하면 디스크가 차단되어 다른 프로세스에서 해당 프로세서를 사용할 수 있습니다.

***** 1번과 2번은 계속해서 바뀔 수 있다 *****
프로세스가 실행중일 수 있다가도 실행될 준비가 된 상태로 반복해서 바뀔 수 있는 것이다.

Q. PCB 란?

PCB (Process Control Block)

- A C-structure that contains information about each process

(각 프로세스에 대한 정보를 포함하는 C 구조)

Register context : a set of registers that define the state of a process

(프로세스의 상태를 정의하는 레지스터들의 집합을 의미)

따라서, OS가 프로세스 정보를 구조체에 저장하고 추적하는 것이 PCB의 핵심이다.