

3. 운영체제

▼ 목차

3-1. 운영체제와 컴퓨터

3-1-1. 운영체제의 역할과 구조

3-1-2. 컴퓨터의 요소

CPU

3-2. 메모리

3-2-1. 메모리 계층

3-2-2. 캐시

3-2-2. 메모리 관리

3-2-3. 메모리 할당

3-2-3. 페이지 교체 알고리즘

3-3. 프로세스와 스레드

3-3-1. 프로세스와 컴파일 과정

3-3-2. 프로세스의 상태

3-3-3. 프로세스의 메모리 구조

3-3-4. PCB

3-3-5. 멀티프로세싱

3-3-6. 스레드와 멀티스레딩

3-3-7. 공유 자원과 임계 영역

3-3-8. 교착 상태

3-4. CPU 스케줄링 알고리즘

3-4-1. 비선점형 방식

3-4-2. 선점형 방식

운영체제 (Operating System)는 사용자가 컴퓨터를 쉽게 다루게 해주는 인터페이스

한정된 메모리나 시스템 자원을 효율적으로 분배함

윈도우, 리눅스 등

수많은 프로그램들을 하드웨어들이 잘 수행할 수 있게 해주는 역할

3-1. 운영체제와 컴퓨터

하드웨어와 소프트웨어 (유저 프로그램)를 관리하는 일꾼인

운영체제와 CPU, 메모리 등으로 이루어진 컴퓨터를 알아보자

3-1-1. 운영체제의 역할과 구조

- 우선 OS는 하드웨어인 CPU, 메모리, 디스크 등을 잘 동작하게 해주는 SW

- 디스크에 있는 파일이 사용자에 의해 실행이 되고
- 그 파일의 일부가 메모리에 올라가고
- 메모리의 일부의 명령어들이 CPU에서 실행된다

운영체제의 역할

운영체제의 역할은 크게 네 가지가 있다

1. CPU 스케줄링과 프로세스 관리

CPU 소유권을 어떤 프로세스에 할당할지, 프로세스의 생성과 삭제, 자원 할당 및 반환을 관리

CPU는 한번에 하나의 프로그램만 실행하는 것이 아니라 윈도우의 수많은 프로그램들이 동시에 (시분할) 실행되고 있다

CPU는 계산만 하기 때문에, CPU가 처리할 일을 가져다 주는 것이 운영체제의 역할

2. 메모리 관리

한정된 메모리를 어떤 프로세스에 얼마나 할당해야 하는지 관리

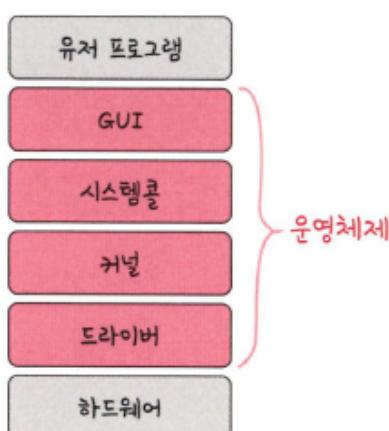
3. 디스크 파일 관리

디스크 파일을 어떤 방법으로 보관할지 관리

4. I/O 디바이스 관리

I/O 디바이스인 마우스, 키보드와 컴퓨터 간에 데이터를 주고 받는 것을 관리

운영체제의 구조



유저 프로그램이 맨 위, 하드웨어가 가장 밑에 있는 구조

GUI, 시스템콜, 커널, 드라이버 부분이 운영체제를 칭한다

<GUI>

사용자가 전자 장치와 상호 작용할 수 있도록 하는 사용자 인터페이스의 한 형태, 단순 명령어 창이 아닌 아이콘을 마우스로 클릭하는 단순한 동작으로 컴퓨터와 상호 작용할 수 있도록 해줌

<드라이버>

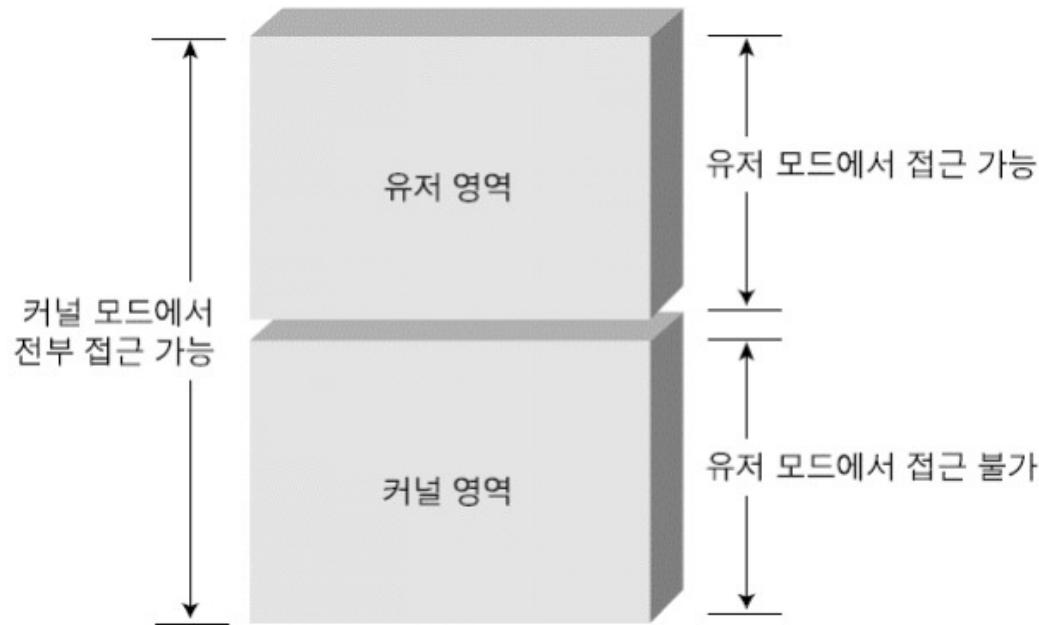
하드웨어를 제어하기 위한 소프트웨어

<CUI>

그래픽이 아닌 명령어로 처리하는 인터페이스

유저 모드와 커널 모드

<시스템의 메모리 영역 분리>



프로그램이 실행되면서 유저모드에서 커널 모드로 전환되기도 하고 커널모드에서 다시 유저 모드로도 전환되기도 한다

우리가 개발하는 프로그램은 일반적으로 유저 모드에서 실행된다

프로그램 실행 중에 인터럽트가 발생하거나 시스템콜을 호출하게 되면 커널 모드로 전환된다

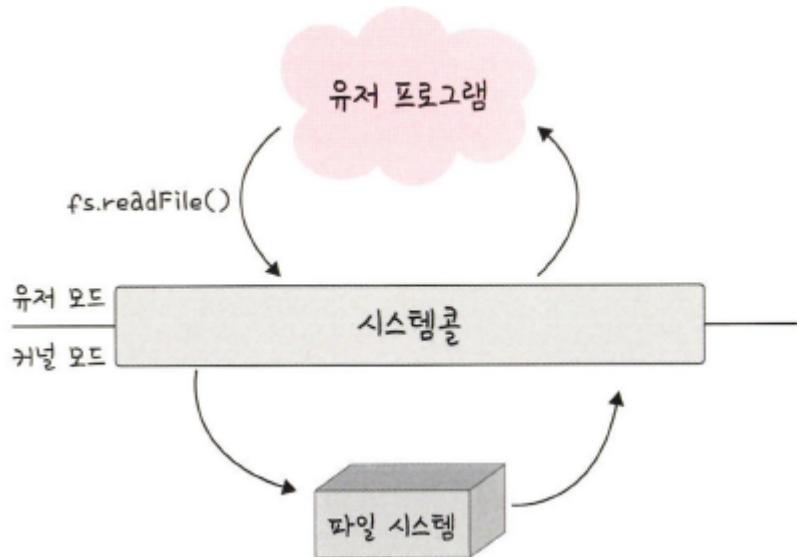
보통 운영체제에서는 커널모드와 유저모드 두가지 프로세서 접근모드를 지원한다. 그 이유는 유저 어플리케이션이 함부로 운영체제의 치명적인 데이터를 수정하거나 삭제하지 못하게 하기 위해서이다. 커널모드는 모든 시스템과 메모리에 접근이 허가된 프로세스 실행 모드이다. 유저모드보다 커널모드에 더 높은 권한을 줌으로써 유저모드에서 에러가 발생했을 때 시스템 전체의 안전성을 보장해 준다.

시스템콜

운영체제가 커널에 접근하기 위한 인터페이스이며 유저 프로그램이 운영체제의 서비스를 받기 위해 커널 함수를 호출할 때 쓴다

- 유저 프로그램이 I/O 요청으로 트랩(trap)을 발동하면
- 올바른 I/O 요청인지 확인한 후
- 유저 모드가 시스템콜을 통해 커널 모드로 변환되어 실행된다.

예를 들어 I/O 요청인 `fs.readFile()` 이라는 파일 시스템의 파일을 읽는 함수가 발동했다고 가정한다면



이때 유저 모드에서 파일을 읽지 않고 커널 모드로 들어가 파일을 읽고 다시 유저 모드로 돌아가 그 뒤에 있는 유저 프로그램의 로직을 수행한다

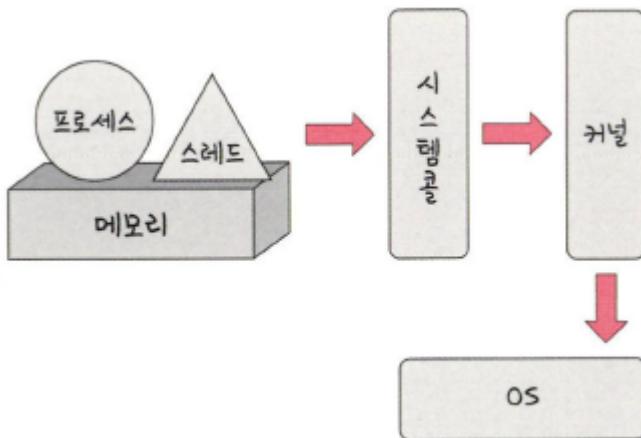
이 과정을 통해 컴퓨터 자원에 대한 직접 접근을 차단할 수 있고, 프로그램을 다른 프로그램으로부터 보호할 수 있다

시스템콜이 필요한 이유

우리가 일반적으로 사용하는 프로그램은 '응용 프로그램'이다. 유저레벨의 프로그램은 유저레벨의 함수들만으로는 많은 기능을 구현하기 힘들기 때문에, **커널(kernel)**의 도움을 반드시 받아야 한다. 이러한 작업은 응용프로그램으로 대표되는 유저 프로세스(User Process)에서 유저모드에서는 수행할 수 없다. 반드시 kernel에 관련된 것은 커널모드로 전환한 후에야, 해당 작업을 수행할 권한이 생긴다.

그렇다면 권한은 왜 필요한 것일까?

그 이유는 만약 권한이 없을 때, 해커가 피해를 입히기 위해 악의적으로 시스템 콜을 사용하는 경우나 초보 사용자가 하드웨어 명령어를 잘 몰라서 아무렇게 함수를 호출했을 경우에 시스템 전체를 망가뜨릴 수도 있기 때문이다. 따라서 이러한 명령어들은 특별하게 **커널 모드**에서만 실행할 수 있도록 설계되었고, 만약 유저 모드에서 시스템 콜을 호출할 경우에는 운영체제에서 불법적인 접근이라 여기고 trap을 발생시킨다.



프로세스나 스레드에서 운영체제로 어떠한 요청을 할 때 시스템콜이라는 인터페이스와 커널을 거쳐 운영체제에 전달된다

이 시스템콜은 하나의 추상화 계층이다. 그렇기 때문에 이를 통해 네트워크 통신이나 데이터베이스와 같은 낮은 단계의 영역 처리에 대한 부분을 많이 신경 쓰지 않고 프로그램을 구현할 수 있는 장점이 있다

modebit

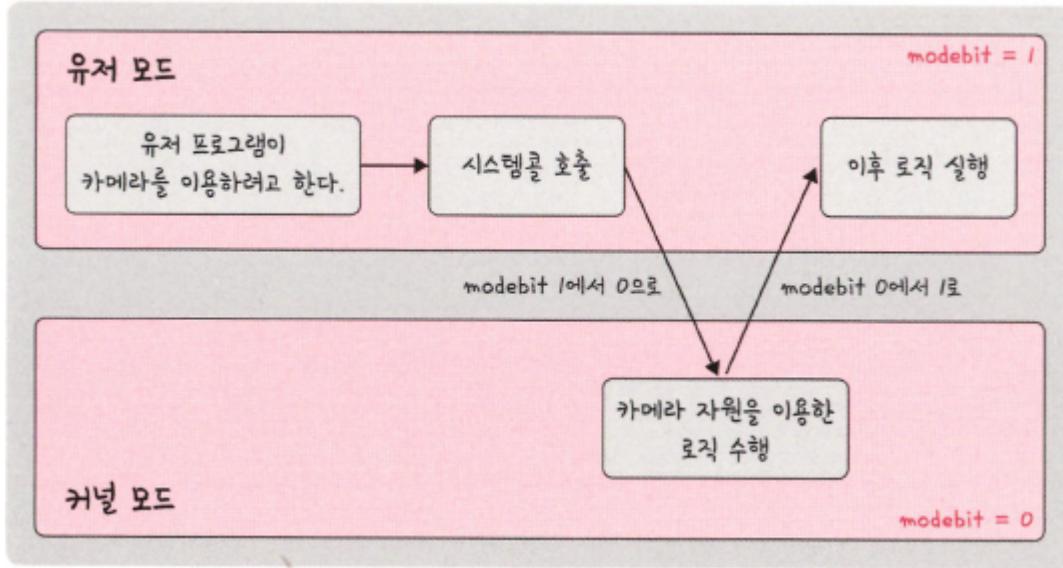
시스템콜이 작동될 때 modebit을 참고해서 유저 모드와 커널 모드를 구분한다

modebit은 1 또는 0 값을 가지는 플래그 변수이다. 카메라, 키보드 등 I/O 디바이스는 운영체제를 통해서만 작동해야 한다.

카메라를 켜는 프로그램이 있다고 가정해보자. 만약 유저 모드를 기반으로 카메라가 켜진다면, 사용자가 의도하지 않았는데 공격자가 카메라를 갑자기 끌 수 있다.

물론 커널 모드를 거쳐 운영체제를 통해 작동한다고 해도 100% 막을 수는 없지만, 운영체제를 통해 작동하게 해야 막기가 쉽다. 이를 위한 장치가 바로 modebit이다.

modebit의 0은 커널모드, 1은 유저 모드라고 설정되며 유저 모드일 경우에는 시스템콜을 못하게 막아서 한정된 일만 가능하게 한다.



- 유저 프로그램이 카메라를 이용할고 할 때 시스템콜을 호출하고
- modebit을 1에서 0으로 바꾸며 커널 모드로 변경한 후 카메라 자원을 이용한 로직을 수행
- 이후에 modebit을 0에서 1로 바꿔서 유저 모드로 변경하고 이후 로직을 수행

<I/O 요청>

입출력 함수, 데이터베이스, 네트워크, 파일 접근 등에 관한 일

<드라이버>

하드웨어를 제어하기 위한 소프트웨어

<유저 모드>

유저가 접근할 수 있는 영역을 제한적으로 두며 컴퓨터 자원에 함부로 침범하지 못하는 모드

<커널 모드>

모든 컴퓨터 자원에 접근할 수 있는 모드

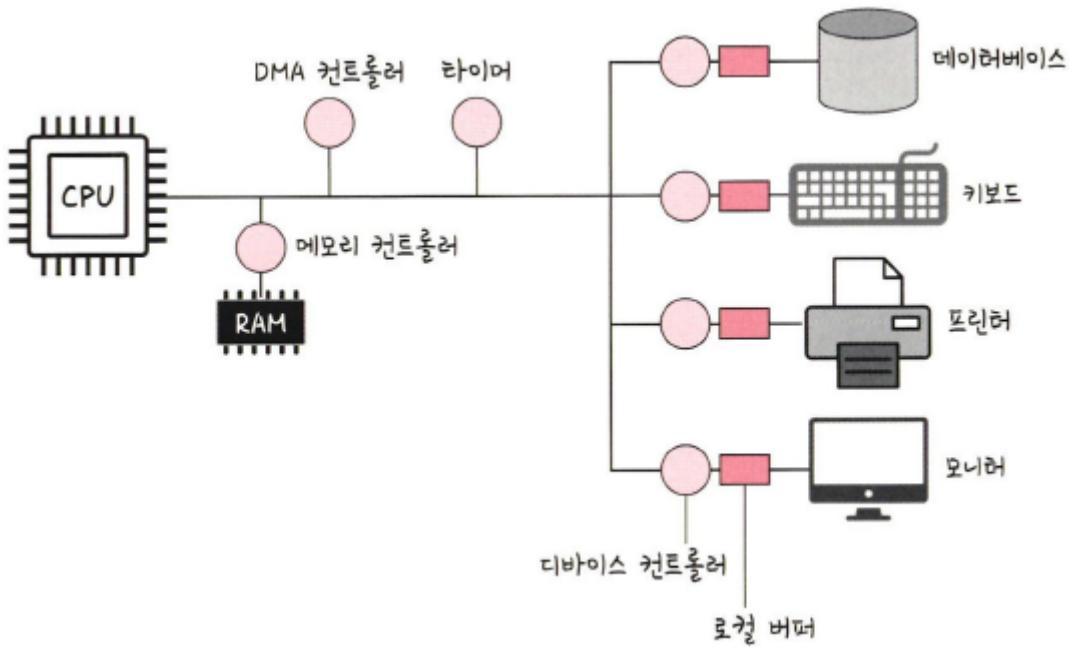
<커널>

운영체제의 핵심 부분이자 시스템콜 인터페이스를 제공

보안, 메모리, 프로세스, 파일 시스템, I/O 디바이스, I/O 요청 관리 등 운영체제의 중추적인 역할

3-1-2. 컴퓨터의 요소

컴퓨터는 CPU, DMA 컨트롤러, 메모리, 타이머, 디바이스 컨트롤러 등으로 이루어짐



CPU

CPU(Central Processing Unit)는 산술논리연산장치, 제어장치, 레지스터로 구성되어 있는 장치

인터럽트에 의해 단순히 메모리에 존재하는 명령어를 해석해서 실행하는 일꾼

관리자 역할을 하는 운영체제의 커널이 프로그램을 메모리에 올려 프로세스로 만들면, 일꾼인 CPU가 이를 처리한다

제어장치

제어장치 (CU, Control Unit)는 프로세스 조작을 지시하는 CPU의 한 부품이다

입출력 장치간 통신을 제어하고 명령어들을 읽고 해석하며 데이터 처리를 위한 순서를 결정

레지스터

CPU 안에 있는 매우 빠른 임시기억장치

CPU와 직접 연결되어 있으므로 연산 속도가 메모리보다 수십 배에서 수백 배까지 빠르다

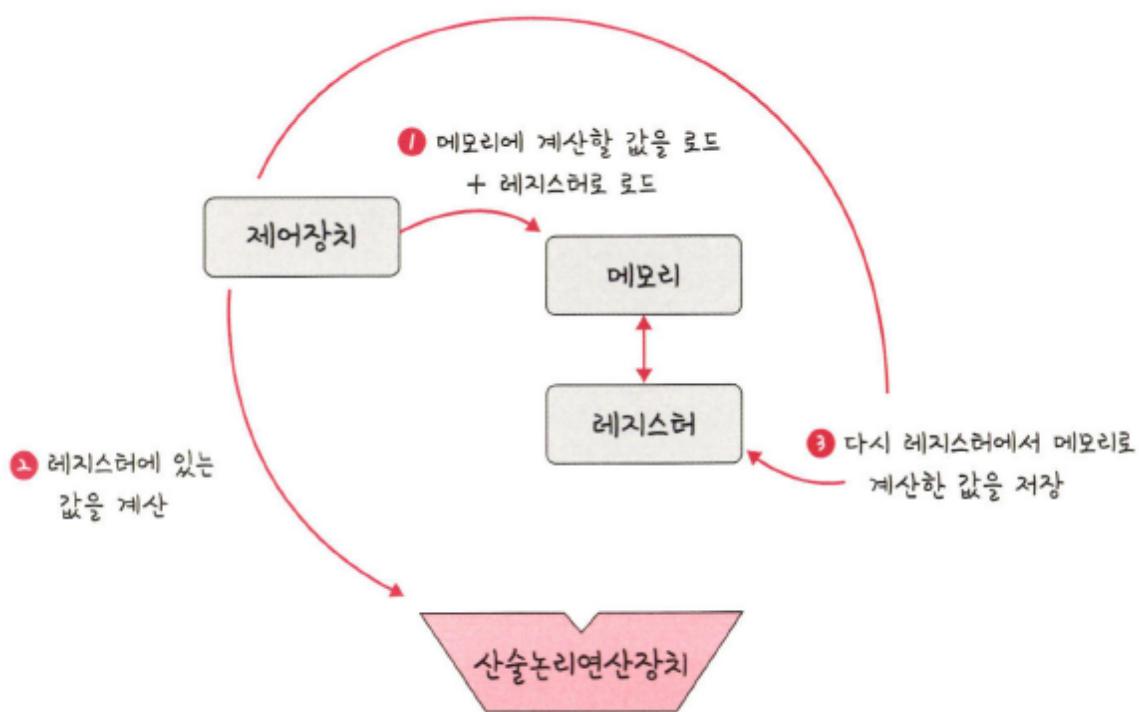
CPU는 자체적으로 데이터를 저장할 방법이 없기 때문에 레지스터를 거쳐 데이터를 전달

산술논리연산장치

산술논리연산장치(ALU, Arithmetic Logic Unit)는 덧셈 뺄셈 같은 두 숫자의 산술 연산과 배타적 논리합, 논리곱 같은 연산을 계산하는 디지털 회로

CPU의 연산 처리

CPU에서 제어장치, 레지스터, 산술논리연산장치를 통해 연산하는 예는 다음과 같다



1. 제어장치가 메모리에 계산할 값을 로드, 레지스터에도 로드
2. 제어장치가 레지스터에 있는 값을 계산하라고 산술논리연산장치에 명령
3. 제어장치가 계산된 값을 다시 레지스터에서 메모리로 계산한 값을 저장

인터럽트

인터럽트는 어떤 신호가 들어왔을 때 CPU를 잠깐 정지시키는 것

키보드, 마우스 등 I/O 디바이스로 인한 인터럽트, 0으로 숫자를 나누는 산술 연산에서의 인터럽트, 프로세스 오류 등으로 발생

인터럽트가 발생되면 인터럽트 핸들러 함수가 모여 있는 인터럽트 백터로 가서 인터럽트 핸들러 함수가 실행된다

인터럽트 간에는 우선순위가 있고 우선순위에 따라 실행되며 인터럽트는 하드웨어 인터럽트, 소프트웨어 인터럽트 두 가지로 나뉜다.

하드웨어 인터럽트

키보드를 연결하거나 마우스를 연결하는 일 등의 I/O 디바이스에서 발생하는 인터럽트를 말함

이때 인터럽트 라인이 설계된 이후 순차적인 인터럽트 실행을 중지하고 운영체제에 시스템 콜을 요청해서 원하는 디바이스로 향해 디바이스에 있는 작은 로컬 버퍼에 접근하여 일을 수

행

소프트웨어 인터럽트

소프트웨어 인터럽트는 **트랩(trap)** 이라고도 한다

프로세스 오류 등으로 프로세스가 시스템콜을 호출할 때 발동

DMA 컨트롤러

DMA 컨트롤러는 IO 디바이스가 메모리에 직접 접근할 수 있도록 하는 하드웨어 장치를 뜻 함

CPU 에만 너무 많은 인터럽트 요청이 들어오기 때문에 CPU 부하를 막아주며 CPU의 일을 부담하는 보조 일꾼이라고 보면 된다

하나의 작업을 CPU와 DMA 컨트롤러가 동시에 하는 것을 방지

메모리

메모리는 전자회로에서 데이터나 상태, 명령어 등을 기록하는 장치를 말한다

보통 RAM을 일컬어 메모리라고 한다. CPU는 계산을 담당하고 메모리는 기억을 담당

CPU - 일꾼

메모리 - 공장의 크기

메모리가 크면 클수록 많은 일을 동시에 진행할 수 있다

타이머

타이머는 몇 초 안에는 작업이 끝나야 한다는 것을 정하고 특정 프로그램에 시간 제한을 다는 역할

시간이 많이 걸리는 프로그램이 작동할 때 제한을 걸기 위해 존재

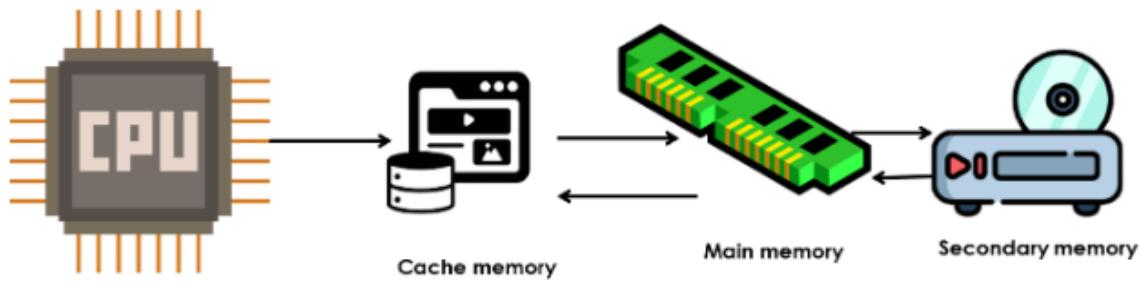
디바이스 컨트롤러

컴퓨터와 연결되어 있는 IO 디바이스들의 작은 CPU를 말한다

3-2. 메모리

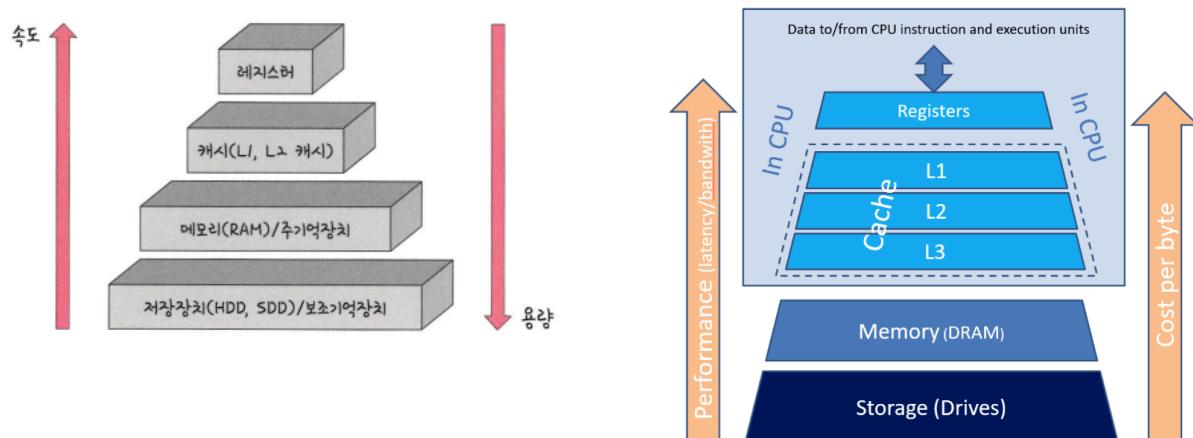
CPU는 그저 메모리에 올라와 있는 프로그램의 명령어들을 실행할 뿐이다

메모리 계층과 메모리 관리에 대해 알아보자



3-2-1. 메모리 계층

메모리 계층은 레지스터, 캐시, 메모리, 저장장치로 구성되어 있다



레지스터

CPU 안에 있는 작은 메모리, 휘발성, 속도 가장 빠름, 기억 용량이 가장 적다

캐시

L1, L2 캐시를 지칭, 휘발성, 속도 빠름, 기억 용량이 적다

주기억장치

RAM을 가리킴, 휘발성, 속도 보통, 기억 용량이 보통

보조기억장치

HDD, SSD가 있다, 휘발성, 속도 낮음, 기억 용량 많다

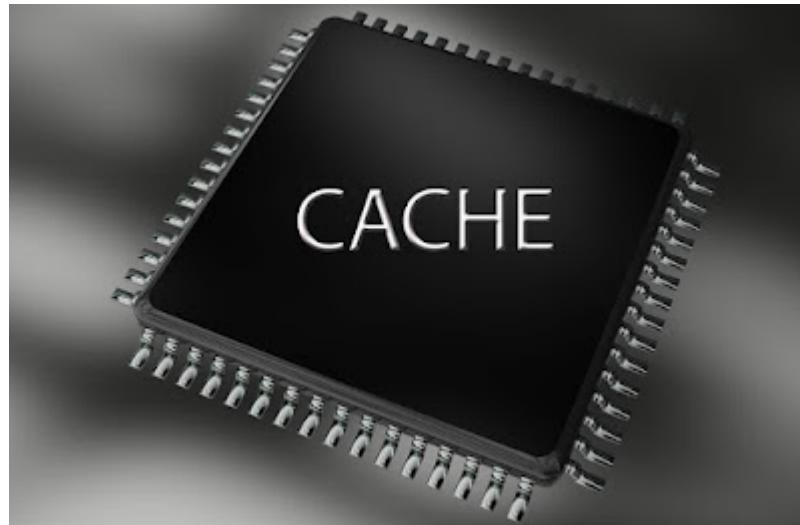
램은 하드디스크로부터 일정량의 데이터를 복사해서 임시로 저장하고 이를 필요 시마다 CPU에 빠르게 전달하는 역할

계층 위로 올라갈수록 가격은 비싸지는데 용량은 작아지고 속도는 빨라진다

이러한 계층이 있는 이유는 경제성과 캐시 때문

3-2-2. 캐시

캐시



캐시는 데이터를 미리 복사해 놓는 임시 저장소이자

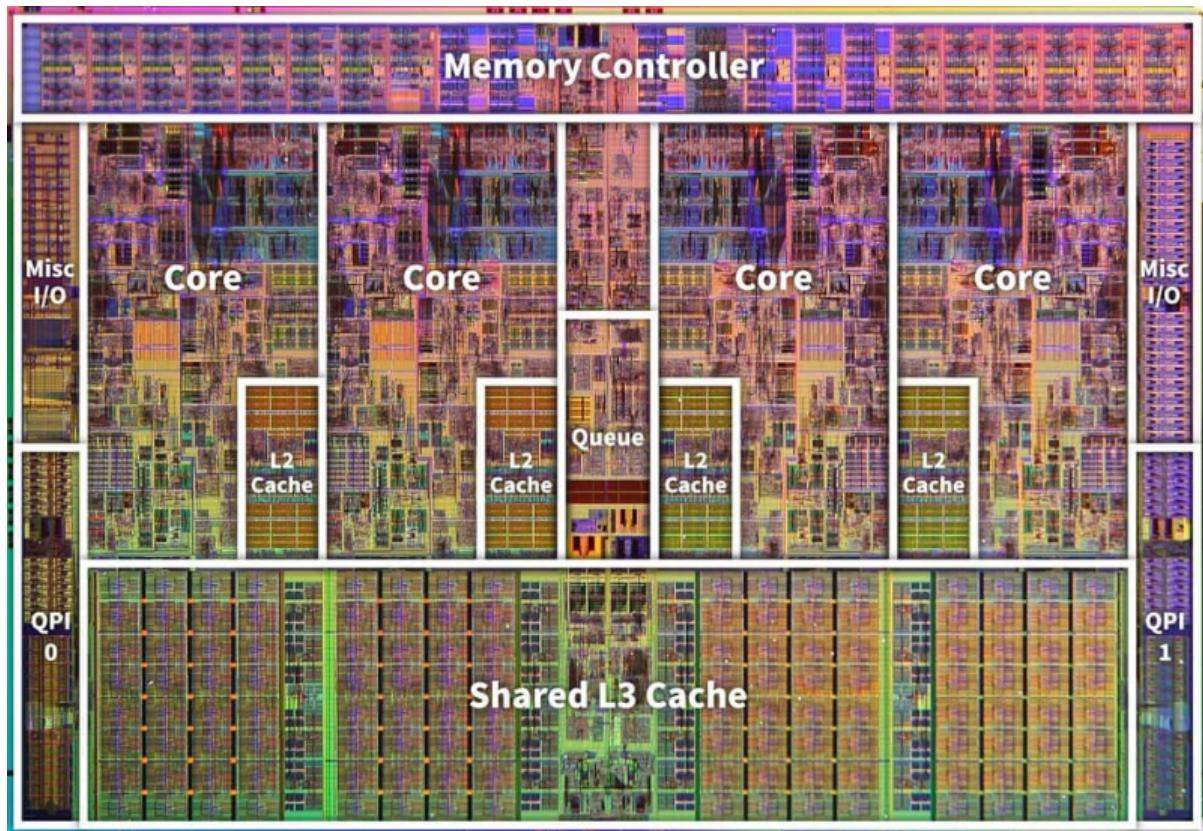
빠른 장치와 느린 장치에서 속도 차이에 따른 병목 현상을 줄이기 위한 메모리를 말한다

이를 통해 데이터를 접근하는 시간이 오래 걸리는 경우를 해결하고 무언가를 다시 계산하는 시간을 절약할 수 있다

실제로 메모리와 CPU 사이의 속도 차이가 너무 크기 때문에 그 중간에 레지스터 계층을 뒤
서 속도 차이를 해결한다

이렇게 속도 차이를 해결하기 위해 계층과 계층 사이에 있는 계층을 캐싱 계층이라고 한다

예를 들어 캐시 메모리와 보조 기억장치 사이에 있는 주기억장치를 보조기억장치의 캐싱 계
층이라고 할 수 있다



- L1 : CPU 내부에 존재 (I-Cache(Instruction Cache) + D-Cache(Data Cache))
 - Cache: 프로세서와 가장 가까운 캐시. 속도를 위해 I\$와 D\$로 나뉜다.
 - Instruction Cache (I\$): 메모리의 TEXT 영역 데이터를 다루는 캐시.
 - Data Cache (D\$): TEXT 영역을 제외한 모든 데이터를 다루는 캐시.
- L2 : CPU와 RAM 사이에 존재
 - 용량이 큰 캐시. 크기를 위해 L1 캐시처럼 나누지 않는다.
- L3 : 보통 메인 보드에 존재
 - 멀티 코어 시스템에서 여러 코어가 공유하는 캐시.

지역성의 원리

캐시 계층을 두는 것 말고 **캐시를 직접 설정하는 방법은?**

이는 **자주 사용하는 데이터를 기반으로 설정해야 한다**

자주 사용하는 데이터에 대한 근거가 되는 것은 지역성이다. 지역성은 시간 지역성과 공간 지역성으로 나뉜다

시간 지역성

최근 사용한 데이터에 다시 접근하려는 특성

예를 들어 for 반복문으로 이루어진 코드 안의 변수 i에 계속해서 접근이 이루어진다. 여기서 데이터는 변수 i이고 최근에 사용했기 때문에 계속 접근해서 +1을 연이어 하는 것을 볼 수 있다.

공간 지역성

최근 접근한 데이터를 이루고 있는 공간이나 그 가까운 공간에 접근하는 특성
코드에서 공간을 나타내는 배열 arr의 각 요소들에 i가 할당되면 해당 배열에 연속적 접근 한다

```
let arr = Array.from({length : 10}, ()=> 0);
console.log(arr)
for (let i = 0; i < 10; i += 1) {
    arr[i] = i;
}
console.log(arr)
/*
[
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0
]
[
    0, 1, 2, 3, 4,
    5, 6, 7, 8, 9
]
*/
```

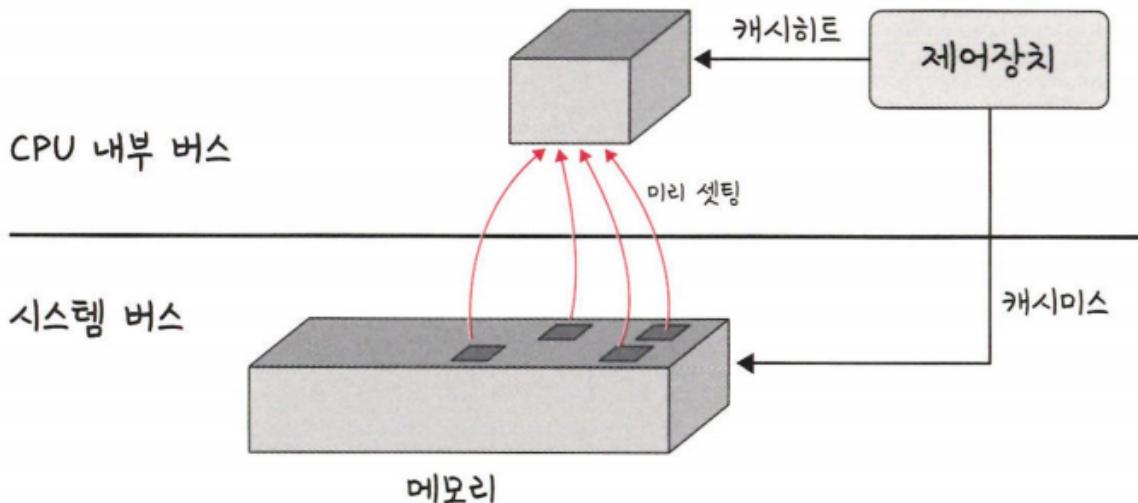
캐시히트와 캐시미스

캐시히트

캐시에서 원하는 데이터를 찾음

캐시미스

해당 데이터가 캐시에 없다면 주메모리로 가서 데이터를 찾아오는 것이 캐시미스



그림처럼 캐시히트를 하게 되면 해당 데이터는 제어장치를 거쳐 오게 된다

캐시히트의 경우 위치도 가깝고 CPU 내부 버스를 기반으로 작동하기 때문에 빠르다

반면 캐시미스가 발생되면 메모리에서 가져오게 되는데, 이는 시스템 버스를 기반으로 작동하기 때문에 느리다

캐시매핑

캐시가 히트되기 위해 매핑하는 방법

CPU의 레지스터와 RAM 간에 데이터를 주고 받을 때를 기반으로 설명한다

레지스터는 RAM에 비해 굉장히 작고 RAM은 굉장히 크기 때문에 작은 레지스터가 캐시 계층으로써 역할을 잘 해주려면 매핑을 어떻게 하는냐가 중요하다

이름	설명
직접 매핑 (directed mapping)	메모리가 1~100이 있고 캐시가 1~10이 있다면 1:1~10, 2:1~20… 이런 식으로 매핑하는 것을 말합니다. 처리가 빠르지만 충돌 발생이 잦습니다.
연관 매핑 (associative mapping)	순서를 일치시키지 않고 관련 있는 캐시와 메모리를 매핑합니다. 충돌이 적지만 모든 블록을 탐색해야 해서 속도가 느립니다.
집합 연관 매핑 (set associative mapping)	직접 매핑과 연관 매핑을 합쳐 놓은 것입니다. 순서는 일치시키지만 집합을 둬서 저장하며 블록화되어 있기 때문에 검색은 좀 더 효율적입니다. 예를 들어 메모리가 1~100이 있고 캐시가 1~10이 있다면 캐시 1~5에는 1~50의 데이터를 무작위로 저장시키는 것을 말합니다.

웹 브라우저의 캐시

**소프트웨어적으로 대표적인 캐시로는 웹 브라우저의 작은 저장소
쿠키, 로컬 스토리지, 세션 스토리지가 있다**

이러한 것들은 보통 사용자의 **커스텀한 정보나 인증 모듈** 관련 사항들을 웹 브라우저에 저장해서 추후 서버에 요청할 때 자신을 나타내는 아이덴티티나 중복 요청 방지를 위해 쓰인다

쿠키

만료기한이 있는 키-값 저장소

4KB 까지 데이터를 저장할 수 있고 만료기한을 정할 수 있다

쿠키를 설정할 때는 `document.cookie`로 쿠리를 볼 수 없게 `httponly` 옵션을 거는 것이 중요
클라이언트 또는 서버에서 만료기한 등을 정할 수 있는데 보통 서버에서 만료기한을 정함

로컬 스토리지

만료기한이 없는 키-값 저장소

10MB 까지 저장할 수 있으며 웹 브라우저를 닫아도 유지되고 도메인 단위로 저장 생성된다
HTML5 를 지원하지 않는 웹 브라우저에서는 사용할 수 없으며 클라이언트에서만 수정 가능

세션 스토리지

만료기한이 없는 키-값 저장소

탭 단위로 세션 스토리지를 생성하며, 탭을 닫을 때 해당 데이터가 삭제된다.

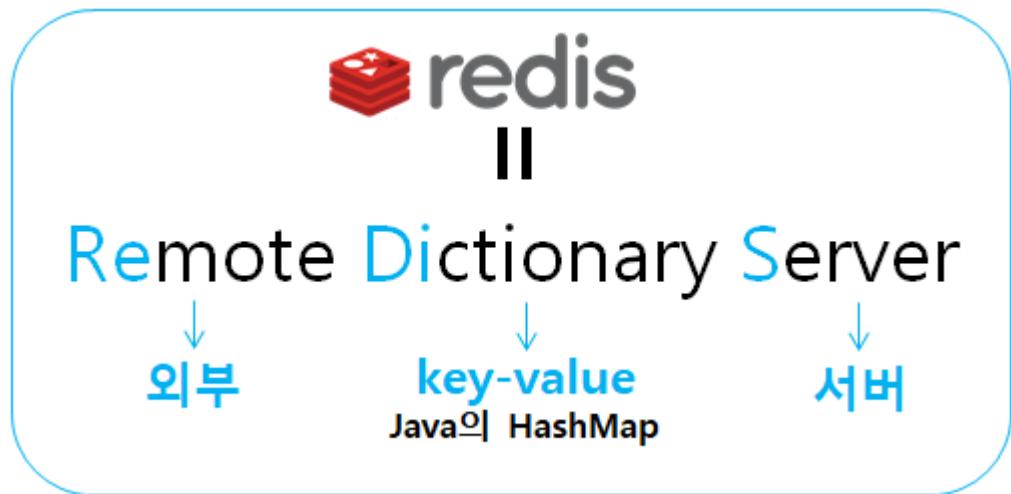
5MB까지 저장이 가능하며 클라이언트에서만 수정 가능

데이터베이스의 캐싱 계층

데이터베이스 시스템을 구축할 때도 메인 데이터 베이스 위에 레디스(redis) 데이터 베이스
계층을 캐싱 계층으로 둘서 성능을 향상 시키기도 한다

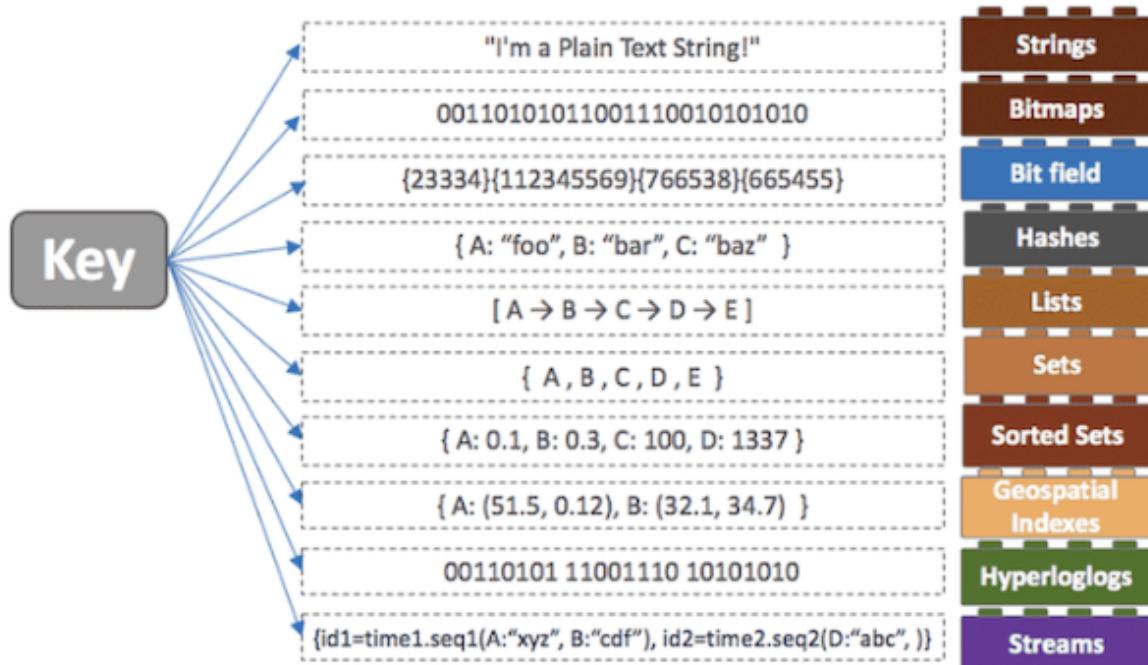
레디스

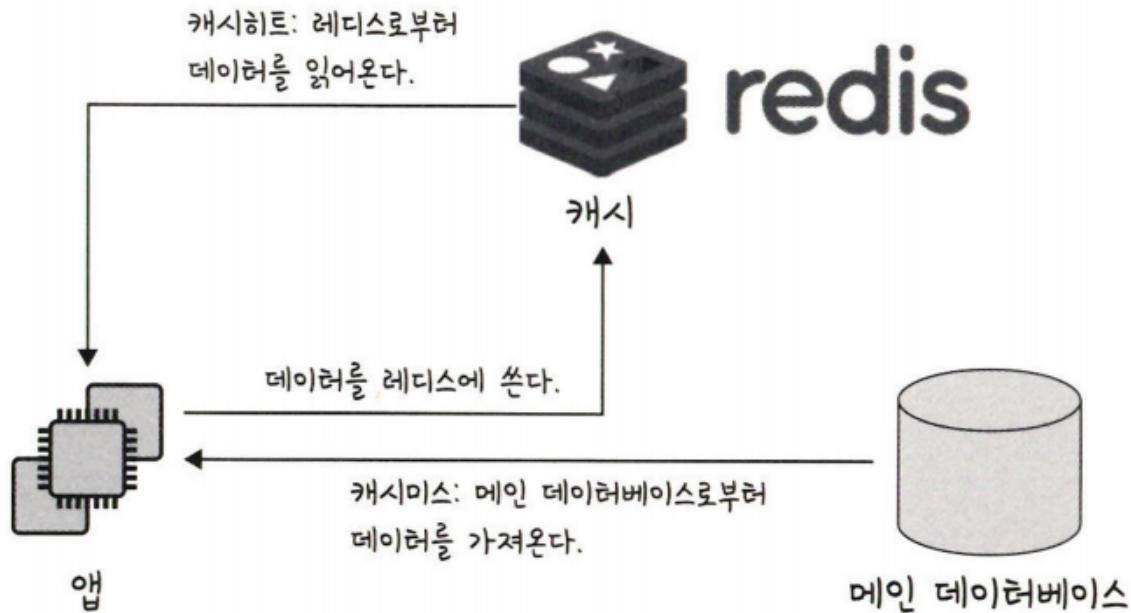
Redis(Remote Dictionary Storage, 레디스)는 모든 데이터를 메모리에 저장하고 조회하는
인메모리 데이터베이스, 메모리 기반의 key-value 구조의 데이터 관리 시스템



● 레디스 대표적인 특징

모든 데이터를 메모리에 저장하고 조회하기 때문에 빠른 Read, Write 속도를 보장하고 또 다양한 자료구조를 지원한다.





3-2-2. 메모리 관리

운영체제의 대표적인 할 일 중 하나가 메모리 관리다
컴퓨터 내의 한정된 메모리를 극한으로 활용해야 한다

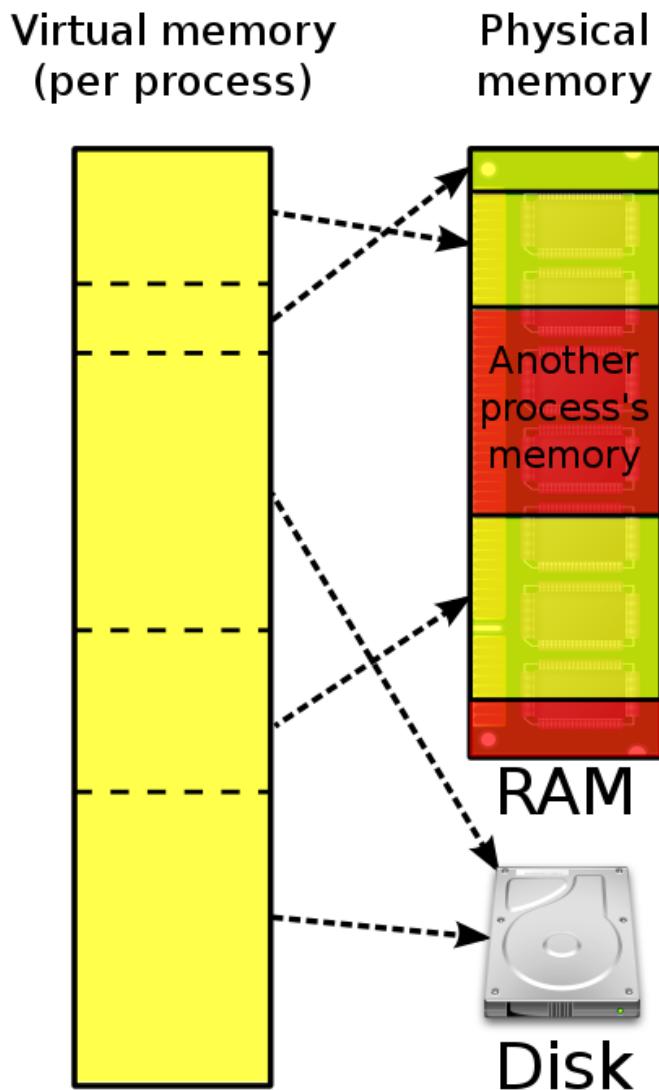
가상 메모리

메모리 관리 기법의 하나

메모리 관리 기법 중 하나로 보조기억(HDD)의 일부를 주기억(RAM)으로 활용하게끔 하여 실제 RAM 보다 큰 메모리 영역을 제공하는 방법으로도 사용된다.

컴퓨터가 실제로 이용 가능한 메모리 자원을 추상화 하여 이를 사용하는 사용자들에게 매우 큰 메모리로 보이게 만드는 것

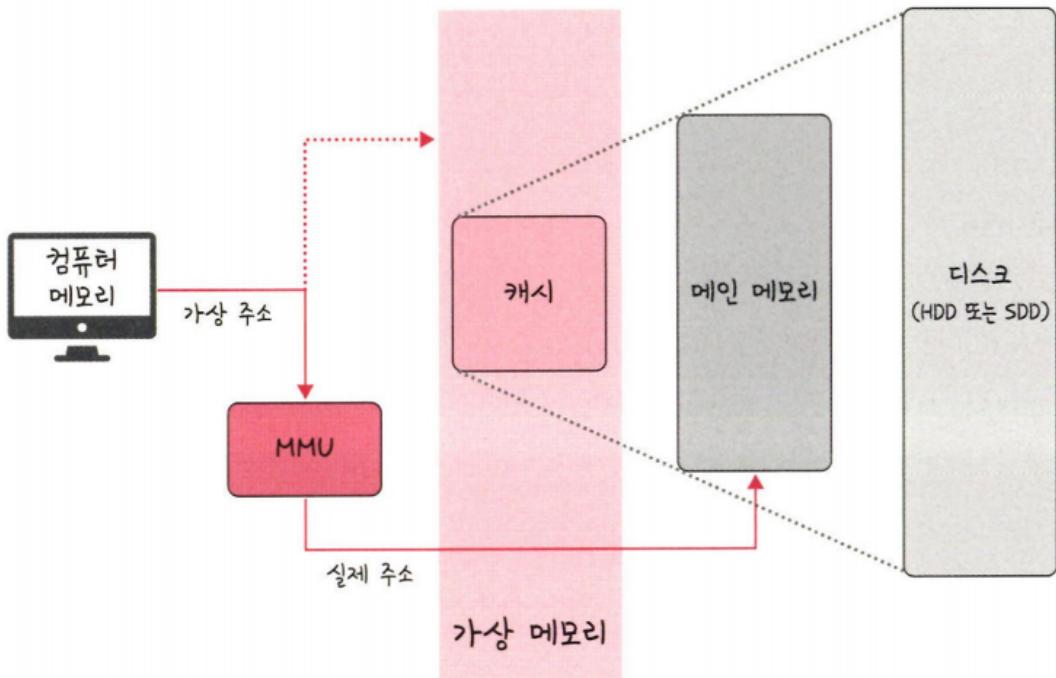
→ 가상 메모리를 사용해 **메인메모리(RAM)의 속도와 하드디스크의 용량의 장점을 가질 수 있다**



<핵심>

- 가상 메모리의 핵심은 필요한 부분만 메모리에 적재(부분적재)하는 것!
1. 프로세스를 실행할 때, 실행에 필요한 부분만 메모리에 올린다 (적재 여부 페어/지터이블에 표시)

실행될 가능성이 낮은 코드
(ex) 오류 처리 코드 등), 과도하게 크게 선언된 데이터 영역(ex) 행렬, 리스트 및 테이블은 실제로 필요한 크기 이상으로 프로그램에서 선언되어 할당됨), 프로그램의 옵션 부분을 처리하는 코드 등은 굳이 메모리에 적재될 필요가 없다!
 2. 애플리케이션의 나머지는 디스크에 남게 됨
 3. 즉 디스크가 RAM의 보조 기억장치처럼 작동



가상적으로 주어진 주소를 가상 주소(logical address) 라고 하며, 실제 메모리상에 있는 주소를 실제 주소 (physical address)라고 한다

가상 주소는 메모리관리장치 (MMU) 에 의해 실제 주소로 변환되며, 이 덕분에 사용자는 실제 주소를 의식할 필요 없이 프로그램을 구축할 수 있게 된다

가상 메모리는 가상 주소와 실제 주소가 매핑되어 있고 프로세스의 주소 정보가 들어 있는 페이지 테이블로 관리된다. 이때 속도 향상을 위해 TLB를 쓴다

TLB

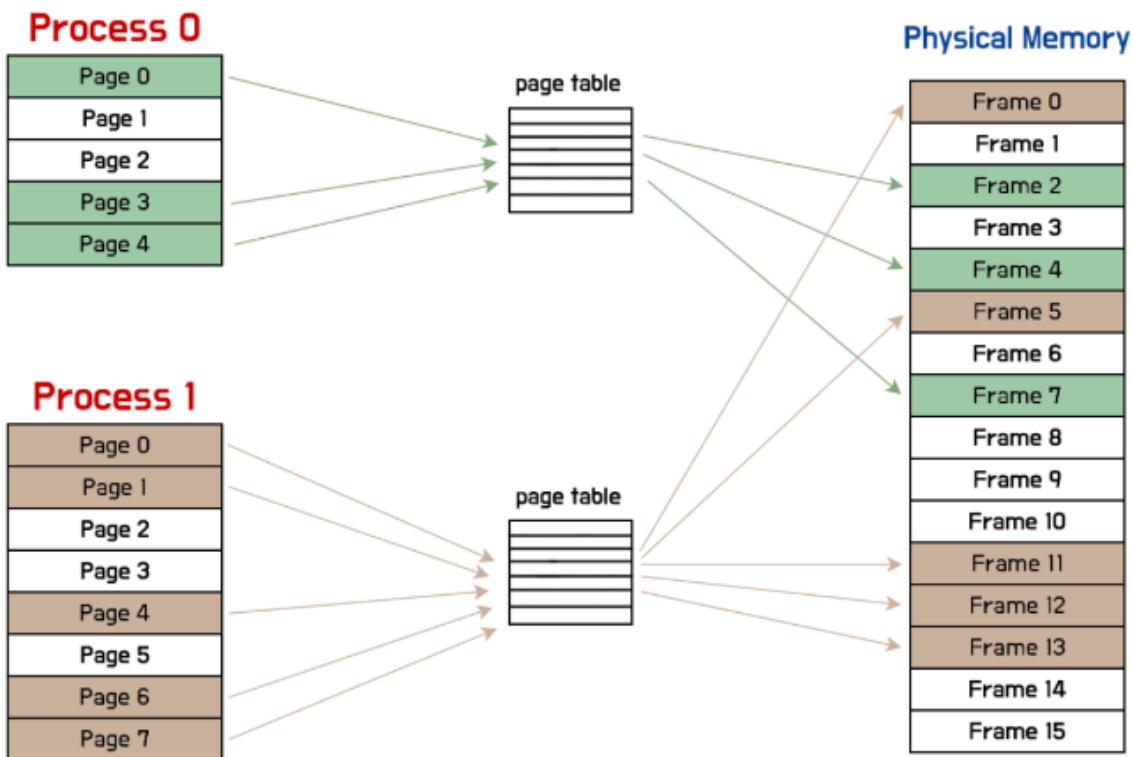
메모리와 CPU 사이에 있는 주소 변환을 위한 캐시.

Page table(os가 관리)

하드 디스크에 옮겨놓은 내용(페이지)들을 사용하려고 할 때 어떻게 참조할것인지 궁금해질 텐데 이는 **Page table**을 사용하여 관리하기 때문에 필요할 때 참조할 수 있다.

page table에는 내용(페이지)들이 저장되어 있는 주소값이 들어있으며 **valid bit**를 사용하여 물리적 메모리에 올라가 있는지 아닌지를 나타낸다.

Virtual address space



스와핑

→ 페이지 폴트를 방지하기 위해 RAM과 하드디스크에 데이터를 올리고 내리고를 반복하는 것

만약 가상 메모리에는 존재하지만 실제 메모리인 RAM에는 현재 없는 데이터나 코드에 접근할 경우, 페이지 폴트가 발생

이를 방지하기 위해 당장 사용하지 않는 영역을 하드디스크로 옮겨 필요할 때 다시 RAM으로 불러와 올리고, 사용하지 않으면 다시 하드디스크로 내림을 반복하여 RAM을 효과적으로 관리하는 것을 스와핑이라고 한다

페이지 폴트

프로세스의 주소 공간에는 존재하지만 지금 컴퓨터의 RAM에는 없는 데이터에 접근했을 경우 발생

이때 운영체제는 다음 과정으로 해당 데이터를 메모리로 가져와서 마치 페이지 폴트가 전혀 발생하지 않은 것처럼 프로그램이 작동하게 해준다

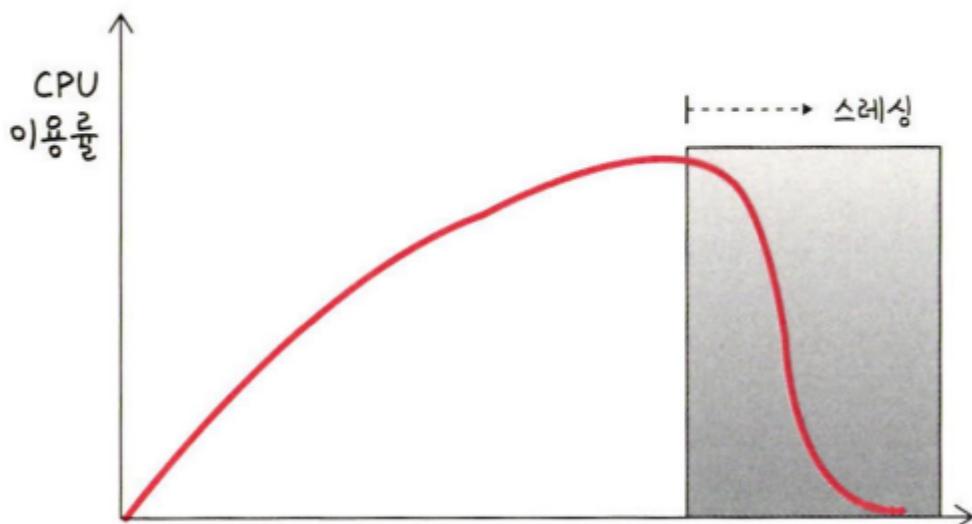
페이지 폴트와 그로 인한 스와핑은 다음 과정으로 이루어짐

1. CPU는 물리 메모리를 확인하여 해당 페이지가 없으면 트랩을 발생해서 운영체제에 알립니다.
2. 운영체제는 CPU의 동작을 잠시 멈춥니다.
3. 운영체제는 페이지 테이블을 확인하여 가상 메모리에 페이지가 존재하는지 확인하고, 없으면 프로세스를 중단하고 현재 물리 메모리에 비어 있는 프레임이 있는지 찾습니다. 물리 메모리에도 없다면 스와핑이 발생됩니다.
4. 비어 있는 프레임에 해당 페이지를 로드하고, 페이지 테이블을 최신화합니다.
5. 중단되었던 CPU를 다시 시작합니다.

페이지 : 가상 메모리를 사용하는 최소 크기 단위, 가상 메모리를 일정한 크기로 나눈 블록
 프레임 : 실제 메모리를 사용하는 최소 크기 단위, 물리 메모리를 일정한 크기로 나눈 블록

스레싱

메모리의 페이지 폴트율이 높은 것을 의미하고 이는 컴퓨터의 심각한 성능 저하를 초래



스레싱은 메모리에 너무 많은 프로세스가 동시에 올라가게 되면 스와핑이 많이 일어나서 발생함

페이지 폴트가 일어나면 CPU 이용률이 낮아진다 (트랩 발생으로 CPU 동작이 일시 정지 되어서?)

CPU 이용률이 낮아지게 되면 운영체제는 CPU가 한가하다고 생각해서 가용성을 더 높이기 위해 더 많은 프로세스를 메모리에 올리게 된다

이러한 악순환이 반복되면 스레싱이 일어나게 된다

이를 해결하기 위한 방법으로는 메모리를 늘리거나, HDD를 SSD로 바꾸기

그 밖에 운영체제에서 이를 해결할 수 있는 방법으로는 작업 세트와 PFF가 있다

작업 세트

프로세스의 과거 사용 이력인 지역성을 통해 결정된 페이지 집합을 만들어서 미리 메모리에 로드하는 것

미리 메모리에 로드하면 탐색에 드는 비용을 줄일 수 있고 스와핑 또한 줄일 수 있다

PFF (Page Fault Frequency)

페이지 폴트 빈도를 조절하는 방법으로 상한선과 하한선을 만드는 방법

만약 상한선에 도달한다면 페이지를 늘리고 하한선에 도달한다면 페이지를 줄이는 것

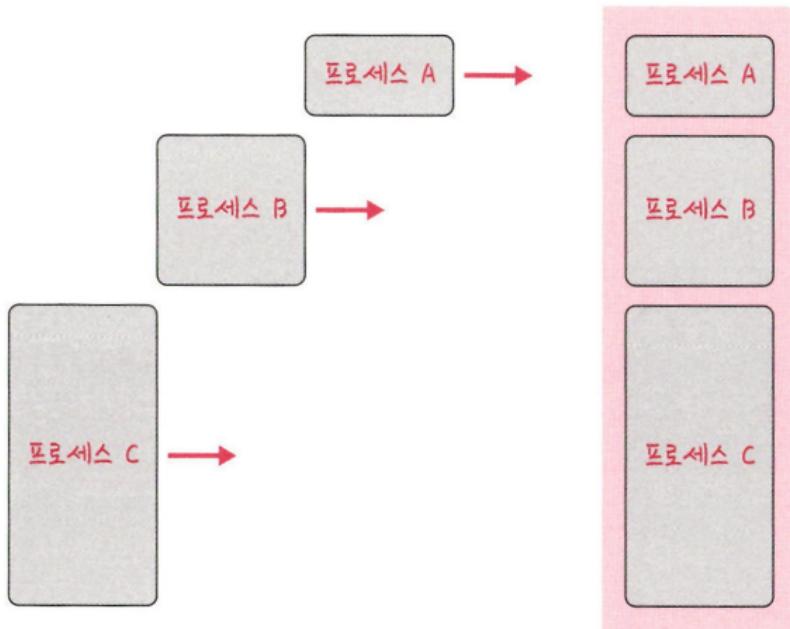
3-2-3. 메모리 할당

메모리에 프로그램을 할당할 때는 시작 메모리 위치, 메모리의 할당 크기를 기반으로 할당하는데,

연속 할당과 불연속 할당으로 나뉜다

연속 할당

메모리에 연속적으로 공간을 할당하는 것



그림처럼 프로세스 A, 프로세스 B, 프로세스 C 가 순차적으로 공간에 할당하는 것을 알 수 있다

이는 메모리를 미리 나누어 관리하는 고정 분할 방식과 매 시점 프로그램의 크기에 맞게 메모리를 분할하여 사용하는 가변 분할 방식이 있다

고정 분할 방식

고정 분할 방식은 메모리를 미리 나누어 관리하는 방식
메모리가 미리 나뉘어 있기 때문에 융통성이 없고 내부 단편화가 발생

가변 분할 방식

매 시점 프로그램의 크기에 맞게 동적으로 메모리를 나눠 사용한다
내부 단편화는 발생하지 않고 외부 단편화는 발생할 수 있다
이는 최초적합, 최저적합, 최악적합이 있다

이름	설명
최초적합	위쪽이나 아래쪽부터 시작해서 훌을 찾으면 바로 할당합니다.
최저적합	프로세스의 크기 이상인 공간 중 가장 작은 훌부터 할당합니다.
최악적합	프로세스의 크기와 가장 많이 차이가 나는 훌에 할당합니다.

용어

— 내부 단편화(internal fragmentation)

메모리를 나눈 크기보다 프로그램이 작아서 들어가지 못하는 공간이 많이 발생하는 현상

— 외부 단편화(external fragmentation)

메모리를 나눈 크기보다 프로그램이 커서 들어가지 못하는 공간이 많이 발생하는 현상, 예를 들어 100MB를 55MB, 45MB로 나눴지만 프로그램의 크기는 70MB일 때 들어가지 못하는 것을 말한다.

— 훌(hole)

할당할 수 있는 비어 있는 메모리 공간이다.

불연속 할당

메모리를 연속적으로 할당하지 않는 불연속 할당은 현대 운영체제가 쓰는 방법으로 불연속 할당인 페이지 기법이 있다

메모리를 동일한 크기의 페이지로 나누고 프로그램마다 페이지 테이블을 두어 이를 통해 메모리에 프로그램을 할당하는 것

페이지 기법 말고도 세그멘테이션, 페이지드 세그멘테이션이 있다

페이지

페이지는 동일한 크기의 페이지 단위로 나누어 메모리의 서로 다른 위치에 프로세스를 할당 한다

훌의 크기가 균일하지 않은 문제가 없어지지만 주소 변환이 복잡해진다

세그멘테이션

세그멘테이션은 페이지 단위가 아닌 의미 단위인 세그먼트로 나누는 방식이다. 프로세스는 코드, 데이터, 스택, 힙 등으로 이루어지는데, 코드와 데이터 등 이를 기반으로 나눌 수도 있으며 함수 단위로 나눌 수도 있음을 의미한다. 공유와 보안 측면에서 좋으며 훌 크기가 균일하지 않은 문제가 발생

페이지드 세그멘테이션

페이지드 세그멘테이션은 공유나 보안을 의미 단위의 세그먼트로 나누고 물리적 메모리는 페이지로 나누는 것을 말함

3-2-3. 페이지 교체 알고리즘

메모리는 한정되어 있기 때문에 스와핑이 많이 일어난다. 스와핑은 많이 일어나지 않도록 설계되어야 하며 이는 페이지 교체 알고리즘을 기반으로 스와핑이 일어난다

오프라인 알고리즘

오프라인 알고리즘은 먼 미래에 참조되는 페이지와 현재 할당하는 페이지를 바꾸는 알고리즘이며, 가장 좋은 방법이다. 그러나 미래에 사용되는 프로세스를 우리는 알 수 없다. 즉 사용할 수 없는 알고리즘이지만 다른 알고리즘과의 성능 비교에 대한 기준을 제공한다

FIFO

가장 먼저 온 페이지를 교체 영역에 가장 먼저 놓는 방법

페이지 프레임 : 3

참조페이지	1	2	3	4	1	2	5	1	2	3	4	5
페이지 프레임	1	1	1	4	4	4	5	5	5	5	5	5
		2	2	2	1	1	1	1	1	3	3	3
			3	3	3	2	2	2	2	2	4	4

프레임이 가득 차면 항상 가장 먼저 들어온 페이지를 내쫓는다.

FIFO는 향후 참조 가능성은 고려하지 않고 그냥 프레임에 올라온 순서대로 내쫓을 대상을 선정한다. 그렇기 때문에 비효율적인 상황이 발생할 수 있다.

LRU (Least Recently Used)

FIFO에서 향후 참조 가능성 조건을 하나 추가한 것이라고 볼 수 있다

LRU는 가장 오랫동안 참조되지 않은 것이 교체대상

LRU는 마지막 참조시점이 가장 오래된 페이지를 내쫓고, 그 자리에 페이지를 넣는 것

→ 오래된 것을 파악하기 위해 각 페이지마다 계수기, 스택을 두어야 하는 문제점이 있다

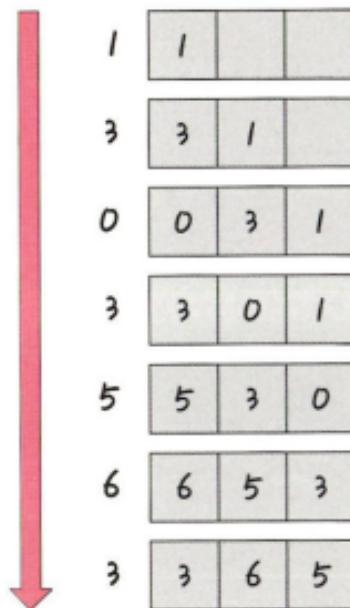
페이지 프레임 : 3

참조페이지	1	2	3	4	1	3	5	3	2	3
페이지 프레임	1	1	1	4	4	4	5	5	5	5
		2	2	2	1	1	1	1	2	2
			3	3	3	3	3	3	3	3

▼ 그림 3-14 LRU

페이지 들어오는 순서 [1,3,0,3,5,6,?]

5번째에서 5번 페이지가 들어왔을 때 가장 오래된 1번 페이지와 스왑하는 것을 볼 수 있는데 이것이 바로 LRU 방식



LRU 구현을 프로그래밍으로 구현할 때는 보통 두 개의 자료구조로 구현한다

→ 해시 테이블과 이중 연결 리스트

해시 테이블은 이중 연결 리스트에서 빠르게 찾을 수 있도록 쓰고, 이중 연결 리스트는 한정된 메모리를 나타냄

C++

코드 위치: ch3/2.cpp

```
#include <bits/stdc++.h>
using namespace std;
class LRUcache {
    list<int> li;
    unordered_map<int, list<int>::iterator> hash;
    int csize;
public:
    LRUcache();
    LRUcache(int);
    void refer(int);
    void display();
};
LRUcache::LRUcache(int n) {
    csize = n;
}
void LRUcache::refer(int x) {
    if (hash.find(x) == hash.end()) {
        if (li.size() == csize) {
            // 가장 끝에 있는 것을 뽑아낸다.
            // 이는 가장 오래된 것을 의미한다.
            int last = li.back();
            li.pop_back();
            hash.erase(last);
        }
    } else {
        li.erase(hash[x]);
    }
    // 해당 페이지를 참조할 때
    // 가장 앞에 넣는다. 또한, 이를 해시 테이블에 저장한다.
    li.push_front(x);
    hash[x] = li.begin();
}
void LRUcache::display() {
    for (auto it = li.begin(); it != li.end(); it++) {
        cout << (*it) << " ";
    }
    cout << "\n";
}
int main() {
    LRUcache ca(3);
    ca.refer(1);
    ca.display();
    ca.refer(3);
    ca.display();
    ca.refer(0);
    ca.display();
    ca.refer(3);
    ca.display();
    ca.refer(5);
    ca.display();
    ca.refer(6);
    ca.display();
    ca.refer(3);
    ca.display();
    ca.refer(5);
    ca.display();
    ca.refer(6);
    ca.display();
    ca.refer(3);
    ca.display();
    return 0;
}
/*
1
3 1
0 3 1
3 0 1
5 3 0
6 5 3
3 6 5
*/

```

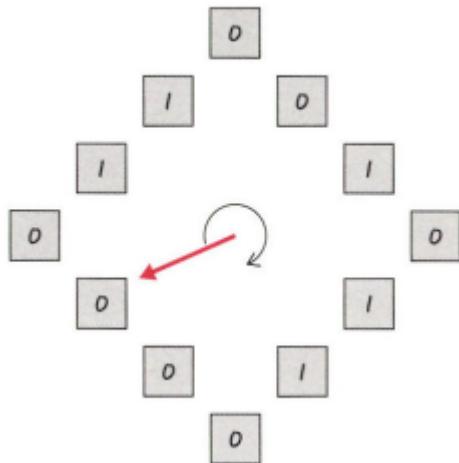
C++로 구현한 코드

해시 테이블을 `unordered_map`으로 구현할 수 있고, 이중 연결 리스트는 `list`로 구현할 수 있다

NUR

LRU에서 발전한 NUR (Not Used Recently) 알고리즘이 있다

▼ 그림 3-15 NUR 알고리즘



clock 알고리즘이라고도 한다

먼저 0과 1을 가진 비트를 둔다. 1은 최근에 참조되었고 0은 참조되지 않음을 의미

시계 방향으로 돌면서 0을 찾은 순간 해당 프로세스를 교체하고, 해당 부분을 1로 바꾼다

LFU (Least Frequently Used)

가장 참조 횟수가 적은 페이지를 교체

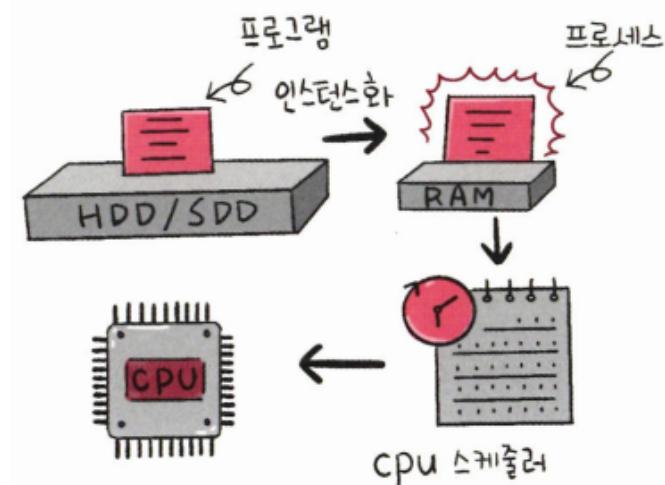
즉 많이 사용되지 않은 것을 교체하는 것이다

3-3. 프로세스와 스레드

프로세스는 컴퓨터에서 실행되고 있는 프로그램

CPU 스케줄링의 대상이 되는 작업이라는 용어와 거의 같은 의미로 쓰임

스레드는 프로세스 내 작업의 흐름



프로그램이 메모리에 올라가고

프로세스가 되는 인스턴스화가 일어나고,

이후 운영체제의 CPU 스케줄러에 따라 CPU가 프로세스를 실행

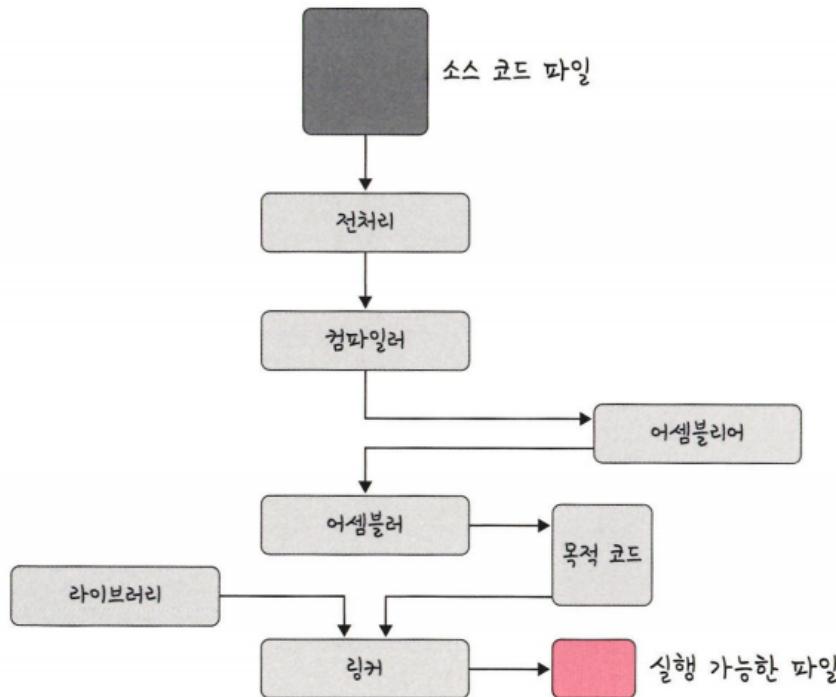
3-3-1. 프로세스와 컴파일 과정

프로그램 —(인스턴스화)—→ 프로세스

프로세스는 프로그램으로부터 인스턴스화된 것을 말함

예를 들어 프로그램은 구글 크롬 프로그램과 같은 실행 파일이며, 이를 두번 클릭하면 구글 크롬 프로세스가 시작되는 것

프로그램은 컴파일러가 컴파일 과정을 거쳐 컴퓨터가 이해할 수 있는 기계어로 번역 실행될 수 있는 파일이 되는 것을 의미하며, '컴파일 과정'이란 다음과 같다



여기서 말하는 프로그램이란 컴파일 과정이 필요한 C언어 기반의 프로그램이다

전처리

소스 코드의 주석을 제거하고 #include 등 헤더 파일을 병합하여 메크로를 치환

컴파일러

오류 처리, 코드 최적화 작업을 하며 어셈블리어로 변환

어셈블러

어셈블리어는 목적 코드로 변환된다. 이때 확장자는 운영체제마다 다른데 리눅스에서는 .o 이다. 예를 들어 가영.c라는 파일을 만들었을 때 가영.o라는 파일이 만들어지게 된다.

링커

프로그램 내에 있는 라이브러리 함수 또는 다른 파일들과 목적 코드를 결합하여 실행 파일을 만든다. 실행 파일의 확장자는 .exe 또는 .out이란 확장자를 갖는다

정적 라이브러리와 동적 라이브러리

라이브러리는 정적 라이브러리와 동적 라이브러리로 나뉜다

정적 라이브러리

프로그램 빌드 시 라이브러리가 제공하는 모든 코드를 실행 파일에 넣는 방식, 시스템 환경 등 외부 의존도가 낮고 코드 중복 등 메모리 효율성이 떨어지는 단점이 있다

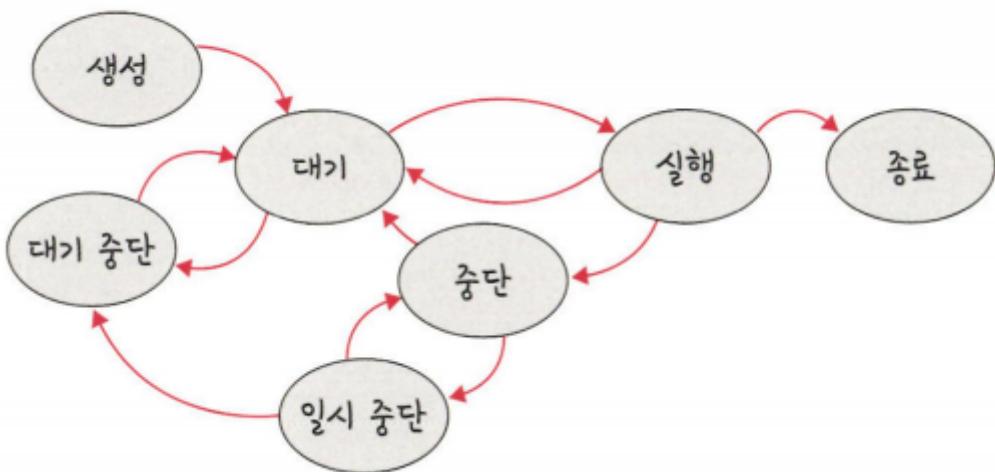
동적 라이브러리

프로그램 실행 시 필요할 때만 DLL이라는 함수 정보를 통해 참조하는 방식, 메모리 효율성에서의 장점과 외부 의존도가 높아진다는 단점

3-3-2. 프로세스의 상태

프로세스의 상태는 여러가지 상태 값을 갖는다

▼ 그림 3-18 프로세스의 상태



생성 상태

프포세스가 생성된 상태를 의미

fork() 또는 exec() 함수를 통해 생성한다. 이때 PCB가 할당된다

fork()

fork()는 부모 프로세스의 주소 공간을 그대로 복사하며, 새로운 자식 프로세스를 생성하는 함수

주소 공간만 복사할 뿐이지 부모 프로세스의 비동기 작업 등을 상속하지 않는다

exec()

exec()은 새롭게 프로세스를 생성하는 함수

대기 상태

대기 상태(ready)는 메모리 공간이 충분하면 메모리를 할당 받고, 아니면 아닌 상태로 대기하고 있으며, CPU 스케줄러로부터 CPU 소유권이 넘어오기를 기다리는 상태

대기 중단 상태

메모리 부족으로 일시 중단된 상태

실행 상태

CPU 소유권과 메모리를 할당 받고 인스트럭션을 수행 중인 상태를 의미

이를 CPU burst가 일어났다고 표현

중단상태

어떤 이벤트가 발생한 이후, 기다리며 프로세스가 차단된 상태

IO 디바이스에 의한 인터럽트로 이런 현상이 많이 발생하기도 한다

예를 들어 프린트 인쇄 버튼을 눌렀을 때, 프로세스가 잠깐 멈춘 듯할 때가 있다

이게 바로 그 상태이다

일시 중단 상태

일시 중단 상태는 대기 중단과 유사

중단된 상태에서 프로세스가 실행되려고 했지만 메모리 부족으로 일시 중단된 상태

종료상태

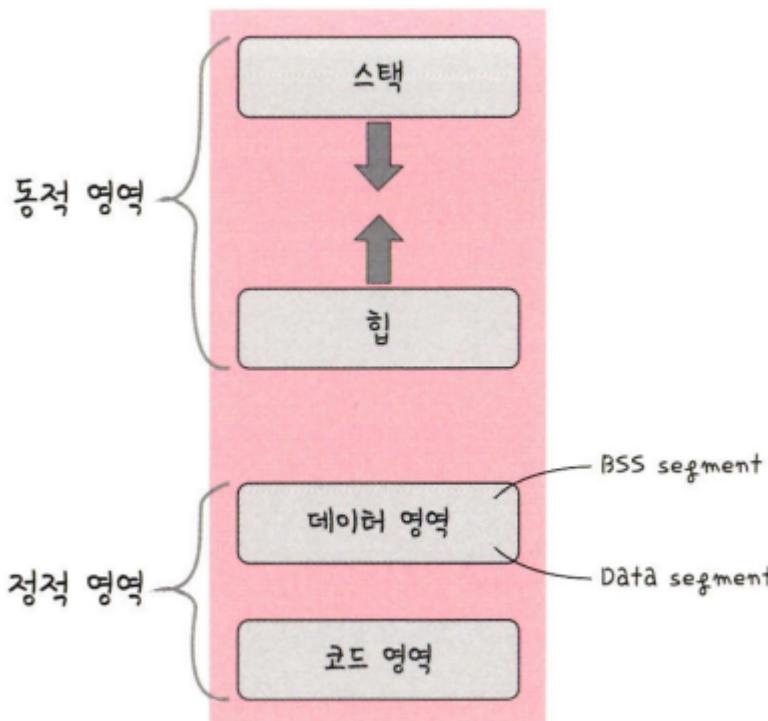
메모리와 CPU 소유권을 모두 놓고 가는 상태

종료는 자연스럽게 종료되는 것도 있지만 부모 프로세스가 자식 프로세스를 강제시키는 비자발적 종료로 종료 되는 것도 있다

자식 프로세스에 할당된 자원의 한계치를 넘어서거나 부모 프로세스가 종료되거나 사용자가 process kill 등 여러 명령어로 프로세스를 종료할 때 발생

3-3-3. 프로세스의 메모리 구조

운영체제는 프로세스에 적절한 메모리를 다음 구조를 기반으로 할당



스택, 힙, 데이터 영역, 코드 영역으로 나눠짐

스택

스택에는 지역변수, 매개변수, 함수가 저장되고 컴파일 시에 크기가 결정되며 '동적'인 특징을 갖는다

스택 영역은 함수가 함수를 재귀적으로 호출하면서 동적으로 크기가 늘어날 수 있는데, 이때 힙과 스택의 메모리 영역이 겹치면 안 되기 때문에 힙과 스택 사이의 공간을 비워 놓는다

힙

동적 할당할 때 사용되며 런타임 시 크기가 결정된다

예를 들어 백터 같은 동적 배열은 당연히 힙에 동적 할당된다

힙은 '동적'인 특징을 가진다

데이터 영역

전역변수, 정적변수가 저장되고, 정적인 특징을 갖는 프로그램이 종료되면 사라지는 변수가 들어 있는 영역

데이터 영역은 BSS 영역과 Date 영역으로 나뉘고, BSS 영역은 초기화가 되지 않는 변수가 0으로 초기화되어 저장되며 Date 영역은 0이 아닌 다른 값으로 할당된 변수들이 저장된다

코드영역

프로그램에 내장되어 있는 소스 코드가 들어가는 영역

이 영역은 수정 불가능한 기계어로 저장되어 있으며, 정적인 특징을 가짐

3-3-4. PCB

PCB(Process Control Block)는 운영체제에서 프로세스에 대한 메타 데이터를 저장한 ‘데이터’를 말하며, 프로세스 제어 블록이라고도 한다.

프로세스가 생성되면 운영체제는 해당 PCB를 생성한다

프로그램이 실행되면 프로세스가 생성되고 프로세스 주소 값들에 앞서 설명한 스택, 힙, 등 구조를 기반으로 메모리가 할당된다. 그리고 이 프로세스의 메타 데이터들이 PCB에 저장되어 관리된다. 이는 프로세스의 중요한 정보를 포함하고 있기 때문에 일반 사용자가 접근하지 못하도록 커널 스택의 가장 앞부분에서 관리된다

→ 메타 데이터

데이터에 관한 구조화된 데이터이자 데이터를 설명하는 작은 데이터,
대량의 정보 가운데에서 찾고 있는 정보를 효율적으로 찾아내서 이용하기 위해 일정한 규칙
에 따라 콘텐츠에 대해 부여되는 데이터이다.

PCB의 구조

PCB는 프로세스 스케줄링 상태, 프로세스 ID 등의 다음과 같은 정보로 이루어짐

- **프로세스 스케줄링 상태:** ‘준비’, ‘일시중단’ 등 프로세스가 CPU에 대한 소유권을 얻은 이후의 상태
- **프로세스 ID:** 프로세스 ID, 해당 프로세스의 자식 프로세스 ID
- **프로세스 권한:** 컴퓨터 자원 또는 I/O 디바이스에 대한 권한 정보
- **프로그램 카운터:** 프로세스에서 실행해야 할 다음 명령어의 주소에 대한 포인터
- **CPU 레지스터:** 프로세스를 실행하기 위해 저장해야 할 레지스터에 대한 정보
- **CPU 스케줄링 정보:** CPU 스케줄러에 의해 중단된 시간 등에 대한 정보
- **계정 정보:** 프로세스 실행에 사용된 CPU 사용량, 실행한 유저의 정보
- **I/O 상태 정보:** 프로세스에 할당된 I/O 디바이스 목록

컨텍스트 스위칭

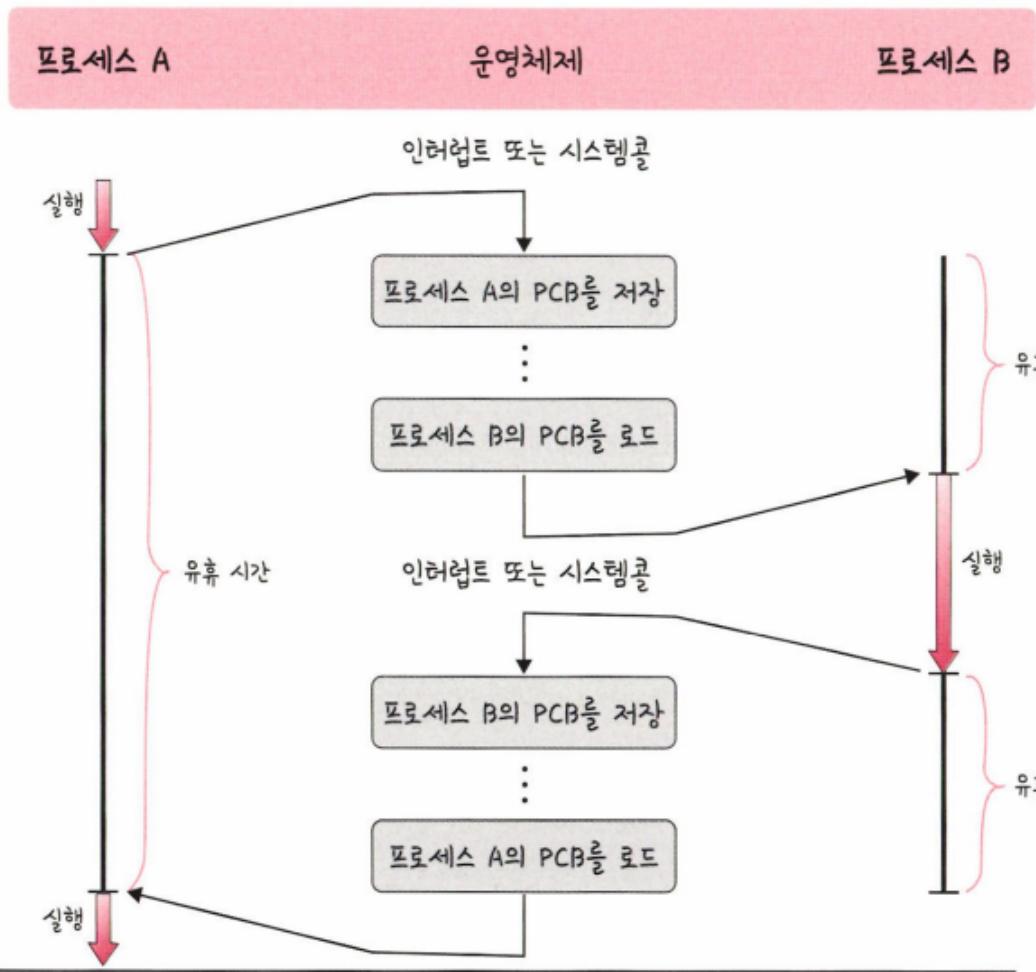
컨텍스트 스위칭은 앞서 설명한 PCB를 교환하는 과정을 말함

한 프로세스에 할당된 시간이 끝나거나 인터럽트에 의해 발생한다

컴퓨터는 많은 프로그램을 동시에 실행하는 것처럼 보이지만 어떠한 시점에서 실행되고 있는 프로세스는 단 한 개이며, 많은 프로세스가 동시에 구동되는 것처럼 보이는 것은 다른 프로세스와의 컨텍스트 스위칭이 아주 빠른 속도로 실행되기 때문

참고로 사실 현대 컴퓨터는 멀티 코어의 CPU를 가지기 때문에 한 시점에서 한 개의 프로그램이라는 설명은 틀린 설명

하지만 컨텍스트 스위칭을 설명할 때는 싱글코어를 기반으로 설명



앞의 그림처럼 한 개의 프로세스 A가 실행하다 멈추고,
프로세스 A의 PCB를 저장하고
다시 프로세스 B를 로드하여 실행
그리고 다시 프로세스 B의 PCB를 저장하고 프로세스 A의 PCB를 로드

컨텍스트 스위칭이 일어날 때 앞의 그림처럼 유회 시간 (idle time)이 발생하는 것을 볼 수 있다. 이뿐만 아니라 이 컨텍스트 스위칭에 드는 비용이 더 있다
그건 바로 캐시 미스

비용 (캐시미스)

컨텍스트 스위칭이 일어날 때 프로세스가 가지고 있는 메모리 주소가 그대로 있으면 잘못된 주소 반환이 생기므로 캐시 클리어 과정을 겪게 되고, 이 때문에 캐시미스가 발생

스레드에서의 컨텍스트 스위칭

참고로 이 컨텍스트 스위칭은 스레드에서도 일어난다

스레드는 스택 영역을 제외한 모든 메모리를 공유하기 때문에 스레드 컨텍스트 스위칭의 경우 비용이 더 적고 시간도 더 적게 걸린다

3-3-5. 멀티프로세싱

멀티프로세싱은 여러 개의 '프로세스', 즉 멀티프로세스를 통해 동시에 두 가지 이상의 일을 수행할 수 있는 것을 말한다

이를 통해 하나 이상의 일을 병렬로 처리할 수 있으며

특정 프로세스의 메모리, 프로세스 중 일부에 문제가 발생 되더라도 다른 프로세스를 이용해서 처리할 수 있으므로 신뢰성이 높은 강점이 있다

웹 브라우저

웹 브라우저는 멀티프로세스 구조를 가지고 있으며 다음과 같다



- **브라우저 프로세스:** 주소 표시줄, 북마크 막대, 뒤로 가기 버튼, 앞으로 가기 버튼 등을 담당하며 네트워크 요청이나 파일 접근 같은 권한을 담당합니다.
- **レン더러 프로세스:** 웹 사이트가 '보이는' 부분의 모든 것을 제어합니다.
- **플러그인 프로세스:** 웹 사이트에서 사용하는 플러그인을 제어합니다.
- **GPU 프로세스:** GPU를 이용해서 화면을 그리는 부분을 제어합니다.

IPC (Inter Process Communication)

멀티프로세스는 IPC가 가능하며 IPC는 프로세스끼리 데이터를 주고 받고 공유 데이터를 관리하는 메커니즘을 뜻함

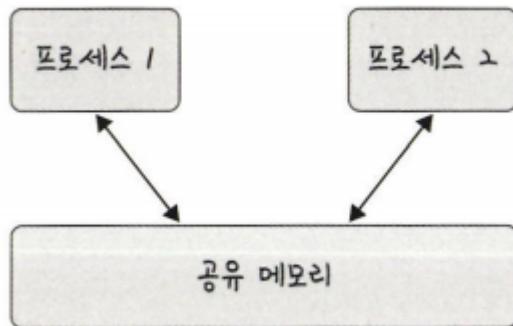
클라이언트와 서버를 예로 들 수 있는데,

클라이언트는 데이터를 요청하고 서버는 클라이언트 요청에 응답하는 것도 IPC

IPC의 종류로는 공유 메모리, 파일, 소켓, 익명 파이프, 명명 파이프, 메시지 큐가 있다. 이들은 모두 메모리가 완전히 공유되는 스레드보다는 속도가 떨어진다

공유 메모리

공유 메모리는 여러 프로세스에 동일한 메모리 블록에 대한 접근 권한이 부여되어 프로세스가 서로 통신할 수 있도록 공유 버퍼를 생성하는 것을 말한다



기본적으로는 각 프로세스의 메모리를 다른 프로세스가 접근할 수 없지만, 공유 메모리를 통해 여러 프로세스가 하나의 메모리를 공유할 수 있다

IPC 방식 중 어떠한 매개체를 통해 데이터를 주고 받는 것이 아닌 메모리 자체를 공유하기 때문에 불필요한 데이터 복사의 오버헤드가 발생하지 않아 가장 빠르며 같은 메모리 영역을 여러 프로세스가 공유하기 때문에 동기화가 필요하다

참고로 하드웨어 관점에서 공유 메모리는 CPU가 접근할 수 있는 큰 랜덤 접근 메모리인 RAM을 가리키기도 한다

파일

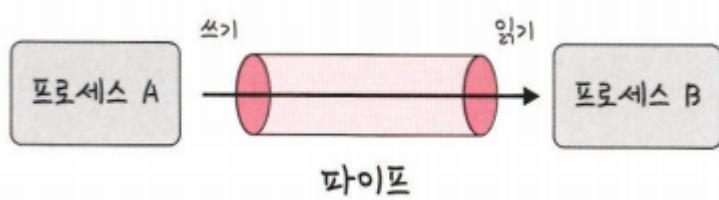
파일은 디스크에 저장된 데이터 또는 파일 서버에서 제공한 데이터를 말한다 이를 기반으로 프로세스 간 통신을 한다

소켓

동일한 컴퓨터의 다른 프로세스나 네트워크의 다른 컴퓨터로 네트워크 인터페이스를 통해 전송하는 데이터를 의미하며 TCP와 UDP가 있다

익명 파이프 (named pipe)

익명 파이프는 프로세스 간에 FIFO 방식으로 읽히는 임시 공간인 파이프를 기반으로 데이터를 주고 받으며, 단방향 방식의 읽기 전용, 쓰기 전용 파이프를 만들어서 작동하는 방식을 말함

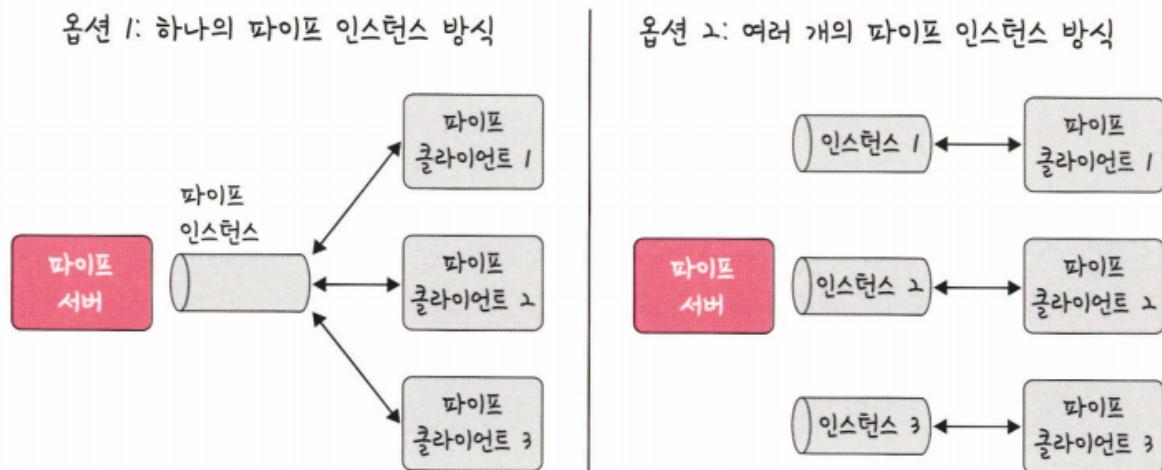


이는 부모, 자식 프로세스 간에만 사용할 수 있으며 다른 네트워크 상에서는 사용이 불가능하다

명명된 파이프

명명된 파이프는 파이프 서버와 하나 이상의 파이프 클라이언트 간의 통신을 위해 명명된 단방향 또는 이중 파이프를 말한다

클라이언트/서버 통신을 위한 별도의 파이프를 제공하며, 여러 파이프를 동시에 사용할 수 있다. 컴퓨터의 프로세스끼리 또는 다른 네트워크상의 컴퓨터와도 통신을 할 수 있다

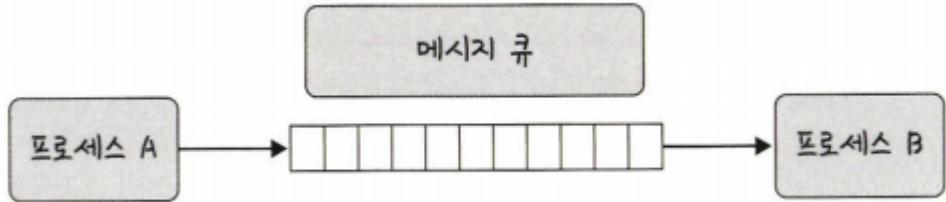


그림처럼 보통 서버용 파이프와 클라이언트용 파이프로 구분해서 작동하며 하나의 인스턴스를 열거나 여러 개의 인스턴스를 기반으로 통신

메시지 큐

메시지 큐는 메시지를 큐 데이터 구조 형태로 관리하는 것을 의미

이는 커널의 전역변수 형태 등 커널에서 전역적으로 관리되며 다른 IPC 방식에 비해서 사용 방법이 매우 직관적이고 간단하며 다른 코드의 수정 없이 단지 몇 줄의 코드를 추가시켜 간단하게 메시지 큐에 접근할 수 있는 장점이 있다



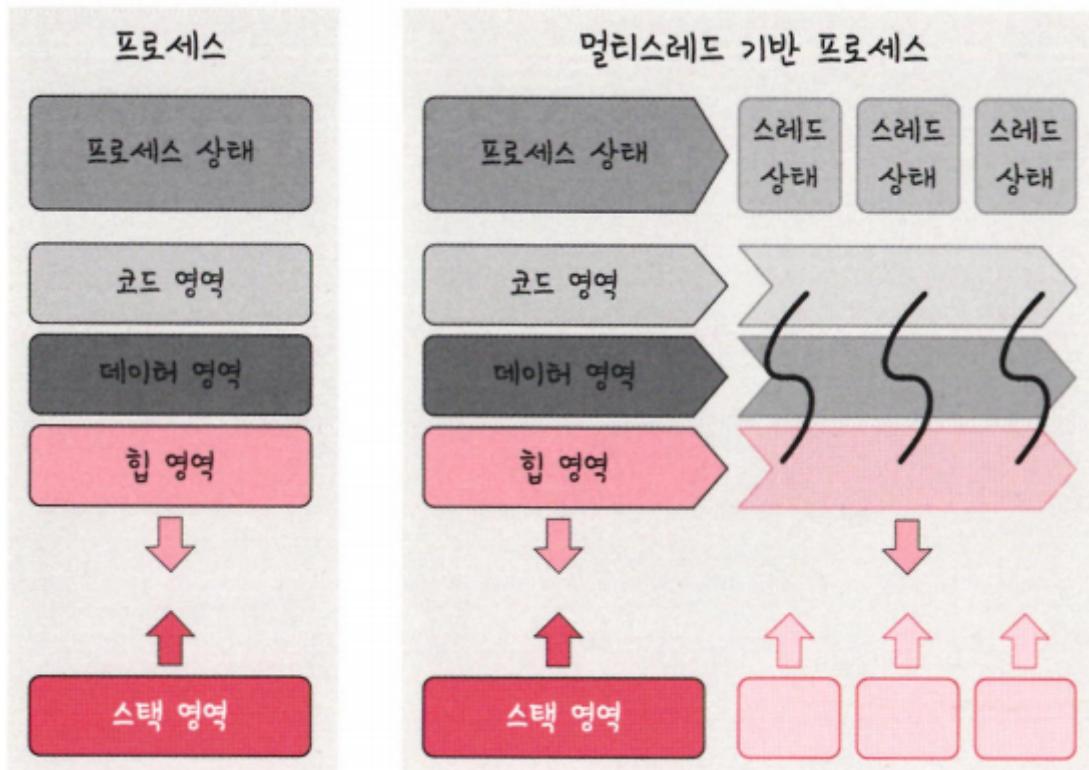
공유 메모리를 통해 IPC를 구현할 때 쓰기 및 읽기 빈도가 높으면 동기화 때문에 기능을 구현하는 것이 매우 복잡해지는데, 이때 대안으로 메시지 큐를 사용하기도 한다

3-3-6. 스레드와 멀티스레딩

스레드

스레드는 프로세스의 실행 가능한 가장 작은 단위

프로세스는 여러 스레드를 가질 수 있다



코드, 데이터, 스택, 힙을 각각 생성하는 프로세스와는 달리 스레드는 코드, 데이터, 힙은 스레드끼리 서로 공유. 그 외의 영역은 각각 생성

멀티스레딩

멀티스레딩은 프로세스 내 작업을 여러 개의 스레드, 멀티스레드로 처리하는 기법이며 스레드끼리 서로 자원을 공유하기 때문에 효율성이 높다

예를 들어 웹 요청을 처리할 때, 새 프로세스를 생성하는 대신 스레드를 사용하는 워커 스레드의 경우 훨씬 적은 리소스를 소비하며, 한 스레드가 중단 되어도 다른 스레드는 실행 상태일 수 있기 때문에 중단되지 않은 빠른 처리가 가능하다

또한 동시성에도 큰 장점이 있다. 하지만 한 스레드에 문제가 생기면 다른 스레드에도 영향을 끼쳐 스레드로 이루어져 있는 프로세스에 영향을 줄 수 있다는 단점

- 동시성

서로 독립적인 작업들을 작은 단위로 나누고 동시에 실행되는 것처럼 보여주는 것

멀티스레스의 예로는 웹 브라우저의 렌더러 프로세스를 들 수 있다



이 프로세스 내에는 메인 스레드, 워커 스레드, 컴포지터 스레드, 레스터 스레드가 존재

3-3-7. 공유 자원과 임계 영역

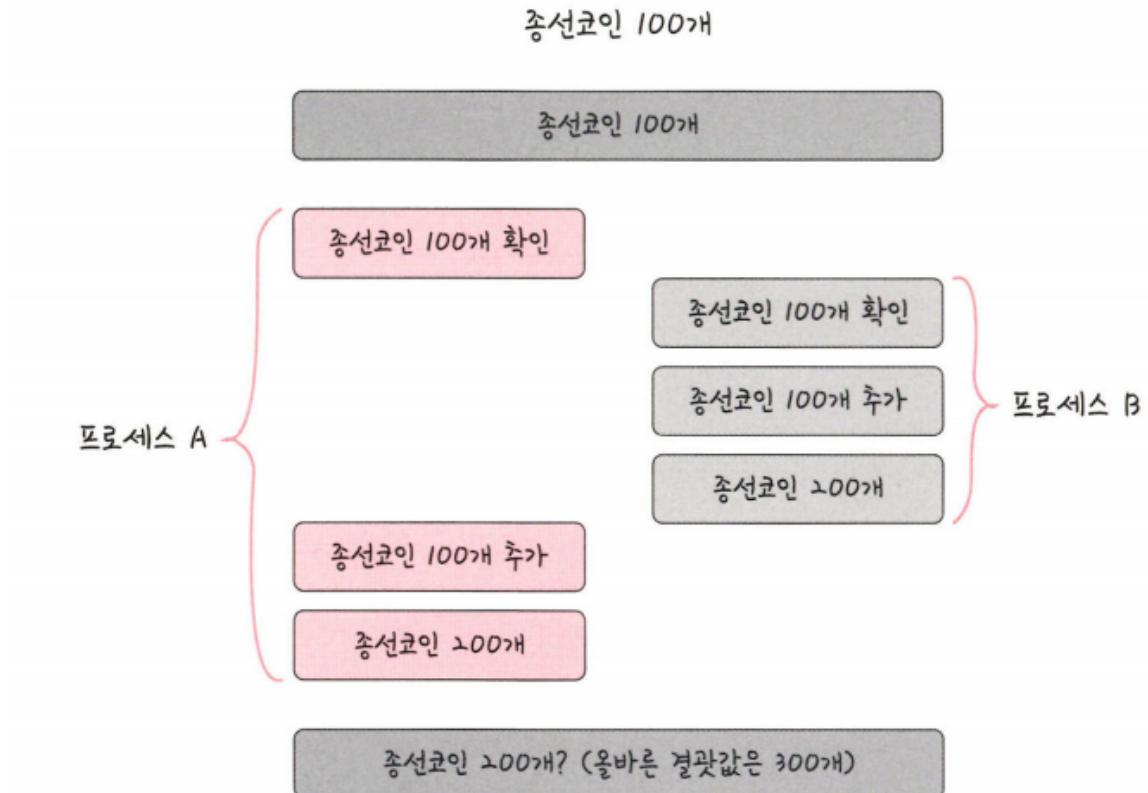
공유자원

공유 자원은 시스템 안에서 각 프로세스, 스레드가 함께 접근할 수 있는 모니터, 프린터, 메모리, 파일, 데이터 등의 자원이나 변수 등을 의미

이 공유 자원을 **두 개 이상의 프로세스가 동시에 읽거나 쓰는 상황을 경쟁 상태**라고 한다
동시에 접근을 시도할 때 접근 타이밍이나 순서 등이 결괏값에 영향을 줄 수 있는 상태인 것

이다

예를 들어 종선코인 100개가 있다고 한다면



프로세스 A와 프로세스 B가 동시에 접근하여 타이밍이 서로 꼬여 정상 결괏값은 300인데 200이 출력 된다

임계 영역

- 공유 자원에 접근할 때 순서 등의 이유로 결과가 달라지는 영역을 임계 영역이라고 한다
- 임계 영역을 해결하기 위한 방법은 크게 뮤텍스, 세마포어, 모니터 세 가지가 있으며, 이 방법 모두 상호 배제, 한정 대기, 융통성이란 조건을 만족한다

이 방법에 토대가 되는 매커니즘은 잠금이다.

예를 들어 임계 구역을 화장실이라고 가정한다면 화장실에 A라는 사람이 들어간 다음 문을 잠근다.

그리고 다음 사람이 이를 기다리다가 A가 나오면 화장실을 쓸 수 있는 것이다.

• 상호 배제

한 프로세스가 임계 영역에 들어갔을 때 다른 프로세스는 들어갈 수 없다

- **한정 대기**

특정 프로세스가 영원히 임계 영역에 들어가지 못하면 안된다

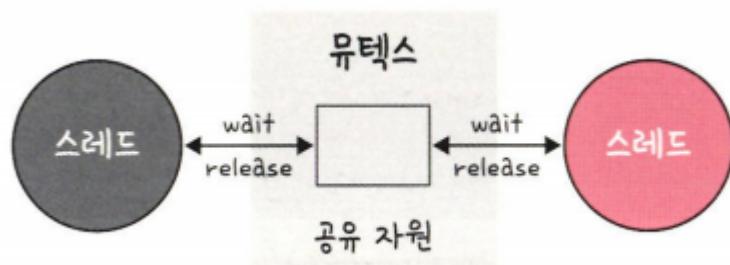
- **유통성**

한 프로세스가 다른 프로세스의 일을 방해해서는 안된다

뮤텍스

공유 자원을 사용하기 전에 설정하고 사용한 후에 해제하는 잠금

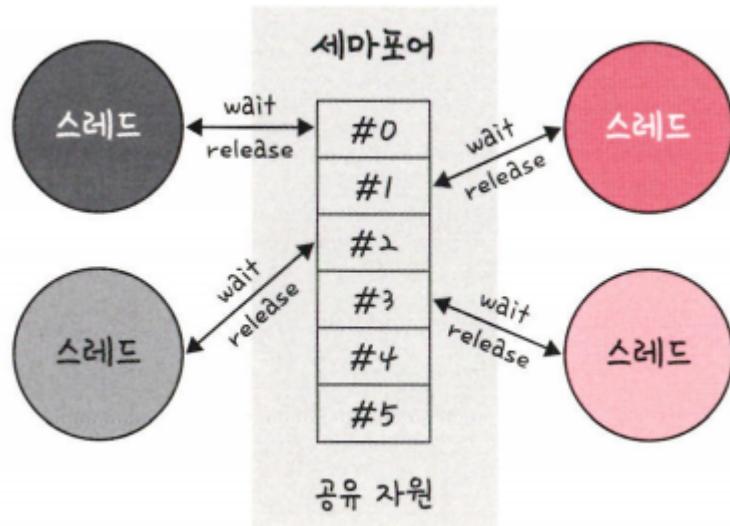
잠금이 설정되면 다른 스레드는 잠긴 코드 영역에 접근 할 수 없다. 또 뮤텍스는 하나의 상태(잠금상태) 만 가진다



세마포어

일반화된 뮤텍스이다. 간단한 정수 값과 두 가지 함수 wait 및 signal로 공유 자원에 대한 접근을 처리한다

wait 는 자신의 차례가 올 때까지 기다리는 함수, signal은 다음 프로세스로 순서를 넘겨주는 함수



프로세스가 공유 자원에 접근하면 세마포어에서 wait 작업을 수행하고

프로세스가 공유 자원을 해제하면 세마포어에서 signal 작업을 수행한다.

세마포어에는 조건 변수가 없고 프로세스가 세마포어 값을 수정할 때 다른 프로세스는 동시에 세마포어 값을 수정할 수 없다.

바이너리 세마포어

바이너리 세마포어는 0과 1의 두 가지 값만 가질 수 있는 세마포어

구현의 유사성으로 인해 뮤텍스는 바이너리 세마포어라고 할 수 있다.

하지만 엄밀히 말하면 뮤텍스는 리소스에 대한 접근을 동기화하는 데 사용되는 잠금 메커니즘이고,

세마포어는 신호를 기반으로 상호 배제가 일어나는 신호 메커니즘이다.

예를 들어 휴대폰에서 노래를 듣다가 친구로부터 전화가 오면 노래가 중지되고 통화 처리 작업에 관한 인터페이스가 등장하는 것

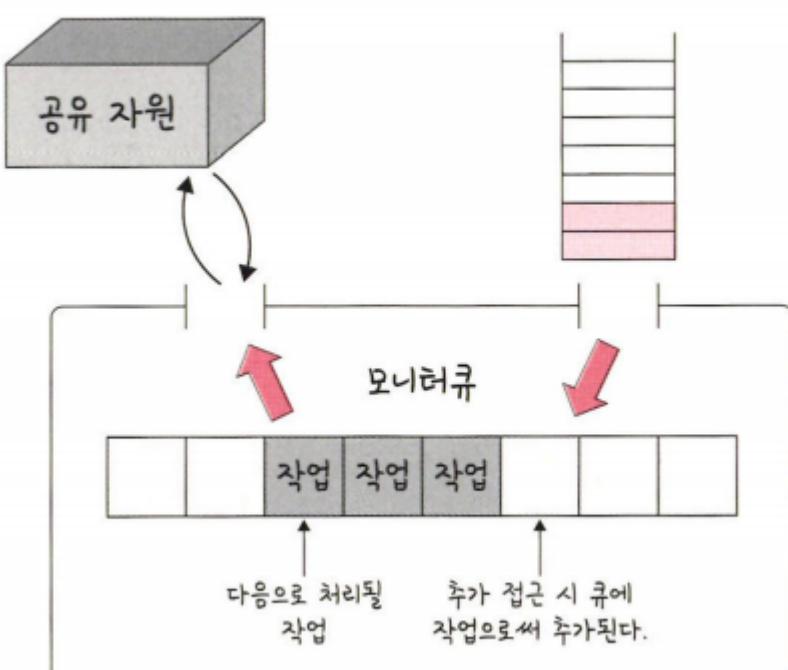
카운팅 세마포어

카운팅 세마포어는 여러 개의 값을 가질 수 있는 세마포어이며, 여러 자원에 대한 접근을 제어하는 데 사용된다.

모니터

모니터는 둘 이상의 스레드나 프로세스가 공유 자원에 안전하게 접근할 수 있도록 공유 자원을 숨기고 해당 접근에 대해 인터페이스만 제공한다

▼ 그림 3-31 모니터

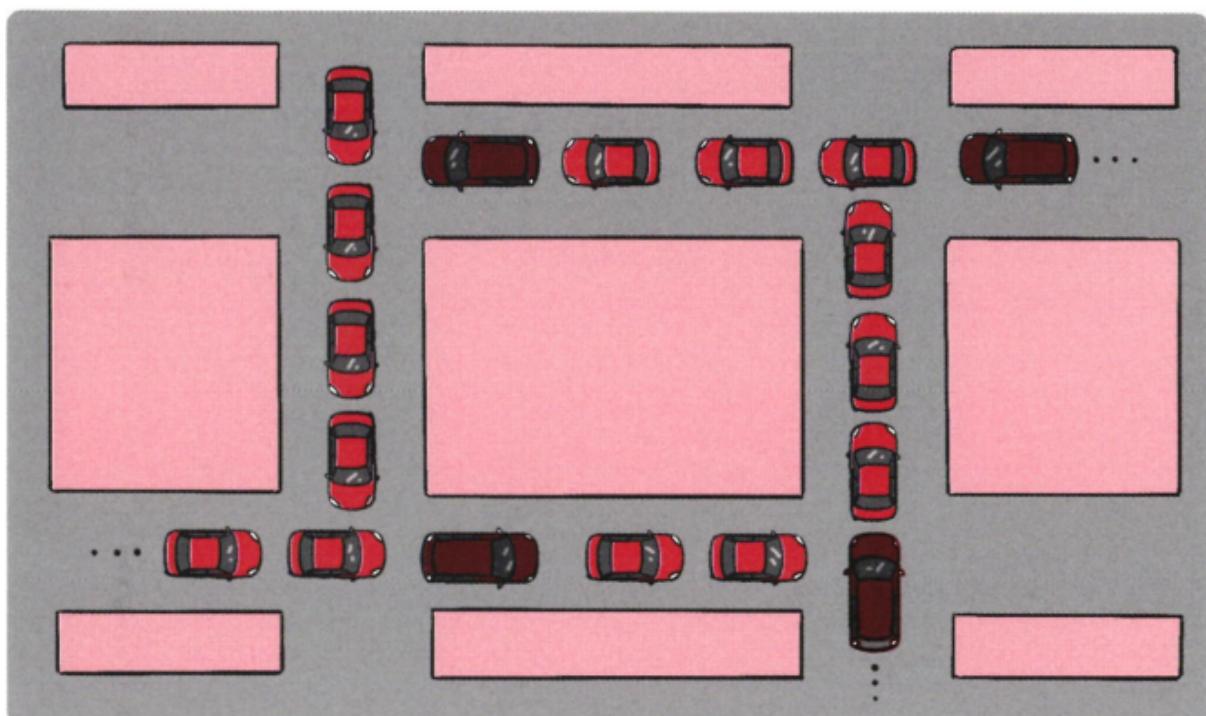


그림처럼 모니터는 모니터 큐를 통해 공유 자원에 대한 작업들을 순차적으로 처리
모니터는 세마포어보다 구현하기 쉬우며 모니터에서 상호 배제는 자동인 반면에,
세마 포어에서는 상호 배제를 명시적으로 구현해야 하는 차이점이 있다.

3-3-8. 교착 상태

교착 상태는 두 개 이상의 프로세스들이 서로가 가진 자원을 기다리며 중단된 상태를 말한다.

예를 들어 프로세스 A가 프로세스 B의 어떤 자원을 요청할 때 프로세스 B도 프로세스 A가 점유하고 있는 자원을 요청하는 것. 이에 대한 원인과 해결 방법은 다음과 같다



교착 상태의 원인

- 상호 배제 : 한 프로세스가 자원을 독점하고 있으며 다른 프로세스들은 접근이 불가능하다
- 점유 대기 : 특정 프로세스가 점유한 자원을 다른 프로세스가 요청하는 상태
- 비선점 : 다른 프로세스의 자원을 강제적으로 가져올 수 없다
- 환형 대기 : 프로세스 A는 프로세스 B의 자원을 요구하고, 프로세스 B는 프로세스 A의 자원을 요구하는 등 서로가 서로의 자원을 요구하는 상황

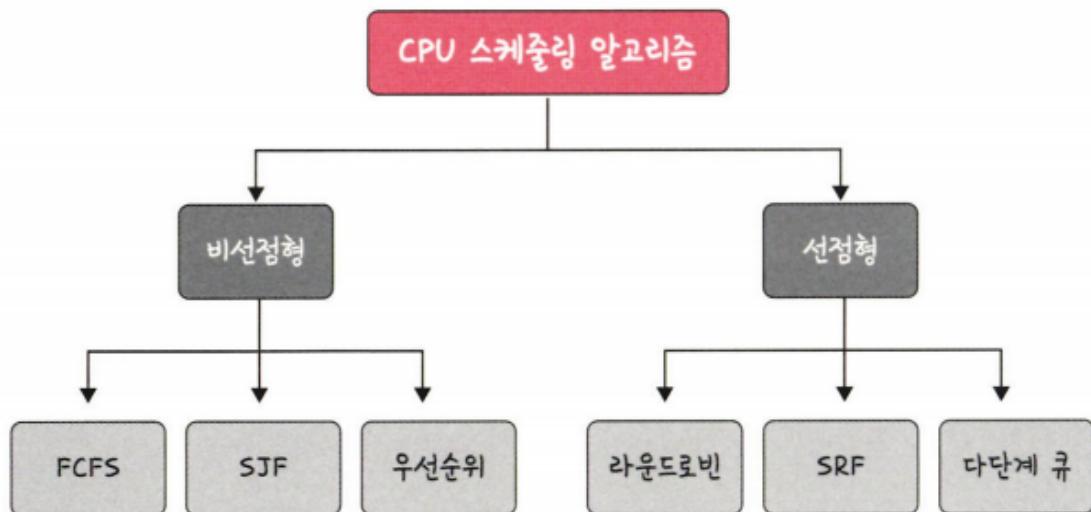
교착 상태 해결 방법

- 자원을 할당할 때 애초에 조건이 성립되지 않도록 설계한다

- 교착 상태 가능성이 없을 때만 자원 할당되며, 프로세스당 요청할 자원들의 최대치를 통해 자원 할당 가능 여부를 파악하는 응행원 알고리즘을 쓴다
- 교착 상태가 발생하면 사이클이 있는지 찾아보고, 이에 관한 프로세스를 한 개씩 지운다
- 교착 상태는 매우 드물게 일어나기 때문에 이를 처리하는 비용이 더 커서 교착 상태가 발생하면 사용자가 작업을 종료한다. 현대 운영체제는 이 방법을 채택했다. 예를 들어 프로세스를 실행시키다 '응답 없음'이라고 뜰 때는 교착 상태가 발생한 경우 이와 같은 경우가 발생하기도 한다.

3-4. CPU 스케줄링 알고리즘

CPU 스케줄러는 CPU 스케줄링 알고리즘에 따라 프로세스에서 해야 하는 일을 스레드 단위로 CPU에 할당



프로그램이 실행될 때는 CPU 스케줄링 알고리즘이 어떤 프로그램에 CPU 소유권을 줄 것인지 결정한다.

이 알고리즘은 CPU 이용률은 높게, 주어진 시간에 많은 일을 하게, 준비 큐에 있는 프로세스는 적게, 응답 시간은 짧게 설정하는 것을 목표로 하고 있다.

3-4-1. 비선점형 방식

프로세스가 스스로 CPU 소유권을 포기하는 방식이며, 강제로 프로세스를 중지하지 않는다
따라서 컨텍스트 스위칭으로 인한 부하가 적다

FCFS

가장 먼저 온 것을 가장 먼저 처리하는 알고리즘

길게 수행되는 프로세스 때문에 '준비 큐에서 오래 기다리는 현상'이 발생한다는 단점이 있다

SJF

실행 시간이 가장 짧은 프로세스를 가장 먼저 실행하는 알고리즘

긴 시간을 가진 프로세스가 실행되지 않는 현상이 일어나며 평균 대기 시간이 가장 짧다
하지만 실제로는 실행 시간을 알 수 없기 때문에 과거의 실행했던 시간을 토대로 추측해서 사용

우선순위

기존 SJF 스케줄링의 경우 긴 시간을 가진 프로세스가 실행되지 않는 현상이 있었다
이는 오래된 작업일수록 '우선순위를 높이는 방법'을 통해 단점을 보완한 알고리즘을 말한다.

3-4-2. 선점형 방식

선점형 방식은 현대 운영체제가 쓰는 방식으로 지금 사용하고 있는 프로세스를 알고리즘에 의해 중단시켜 버리고 강제로 다른 프로세스에 CPU 소유권을 할당하는 방식을 말한다

라운드 로빈

현대 컴퓨터가 쓰는 스케줄링인 우선순위 스케줄링의 일종으로 각 프로세스는 동일한 할당 시간을 주고 그 시간 안에 끝나지 않으면 다시 준비 큐의 뒤로 가는 알고리즘

예를 들어 q 만큼의 할당 시간이 부여되었고 N 개의 프로세스가 운영된다고 하면 ($N - 1$)

* q 시간이 지나면 자기 차례가 오게 됩니다. 할당 시간이 너무 크면 FCFS가 되고 짧으면 컨텍스트 스위칭이 잦아져서 오버헤드, 즉 비용이 커집니다. 일반적으로 전체 작업 시간은 길어지지만 평균 응답 시간은 짧아진다는 특징이 있습니다.

또한, 이 알고리즘은 로드밸런서에서 트래픽 분산 알고리즘으로도 쓰입니다.

SRF

SJF는 중간에 실행 시간이 더 짧은 작업이 들어와도 기존 짧은 작업을 모두 수행하고 그 다음 짧은 작업을 이어나가는데, SRF는 중간에 더 짧은 작업이 들어오면 수행하던 프로세스를 중지하고 해당 프로세스를 수행하는 알고리즘

다단계 큐

우선순위에 따른 준비 큐를 여러 개 사용하고, 큐마다 라운드 로빈이나 FCFS 등 다른 스케줄링 알고리즘을 적용한 것을 말한다. 큐 간의 프로세스 이동이 안되므로 스케줄링 부담이 적지만 유연성이 떨어지는 특징이 있다

