



CS

## 3.1 운영체제와 컴퓨터

### 운영체제란?

- 컴퓨터 하드웨어와 응용 프로그램 간의 상호작용을 관리하고 제어하는 역할
  - 컴퓨터와 사용자 사이에서 커뮤니케이션을 도와주는 역할
  - Windows, MacOs, Linux, Unix ....

### 운영체제의 구조



### 운영체제의 역할

#### 1. CPU 스케줄링과 프로세스 관리

- CPU 소유권을 어떤 프로세스에 할당할지

## 2. 메모리 관리

- 사용 가능한 메모리는 한정되어있기에 어떤 프로세스에 메모리를 얼마큼 할당할지 관리

## 3. 디스크 파일 관리

- 디스크 파일을 어떠한 방법으로 보관할지 관리

## 4. I/O 디바이스 관리

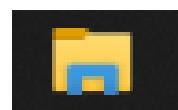
- 마우스, 키보드와 같은 I/O 디바이스와 컴퓨터 간에 데이터를 주고받는 것을 관리

### ▼ 뇌피셜

**사용자가 하드웨어(마우스)로 파일 하나를 사진 폴더로 이동하고 싶다**

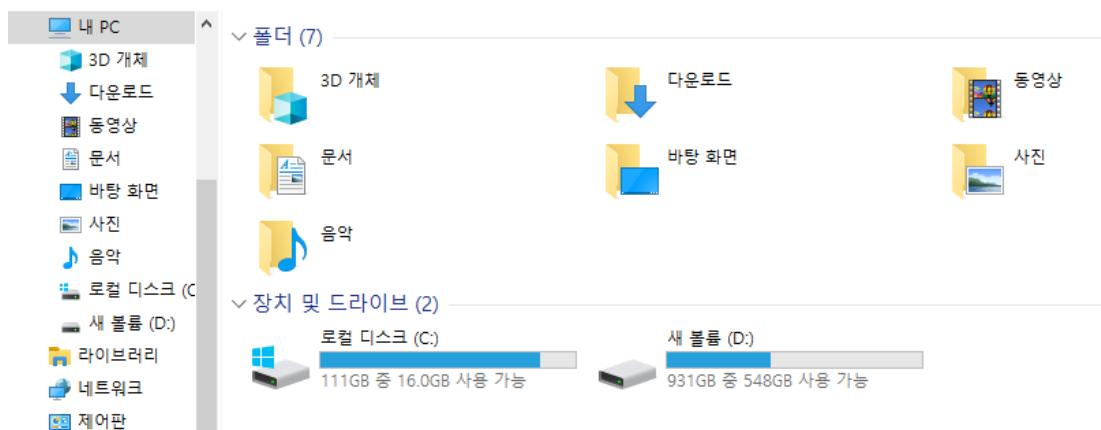
응용 프로그램

- 사용자가 Windows 탐색기를 실행하여 파일 관리를 위한 응용 프로그램을 연다



**GUI(Windows 탐색기의 파일 탐색창)**

- Windows 탐색기는 GUI를 통해 다양한 파일 및 폴더를 시각적으로 제공



## 시스템 콜

- 마우스 이벤트(클릭, 드래그)가 시스템 콜로 변환되어 운영체제에 전달
- 커널 함수 호출

## 커널(파일 시스템 관리)

- 시스템 콜로 전달 받은 이벤트 처리
- 파일 시스템 드라이버를 호출하여 파일을 목적지 폴더로 이동

## 드라이버(파일 시스템 드라이버)

- 파일 시스템 **드라이버가 실제 디스크와 통신**하여 파일을 이동하거나 복사하는 등의 작업을 디스크에 적용

---

## 하드웨어(마우스)의 이벤트(클릭, 드래그)

- 사용자가 마우스를 사용하여 파일을 클릭, 다른 폴더로 드래그

## 하드웨어(디스크)

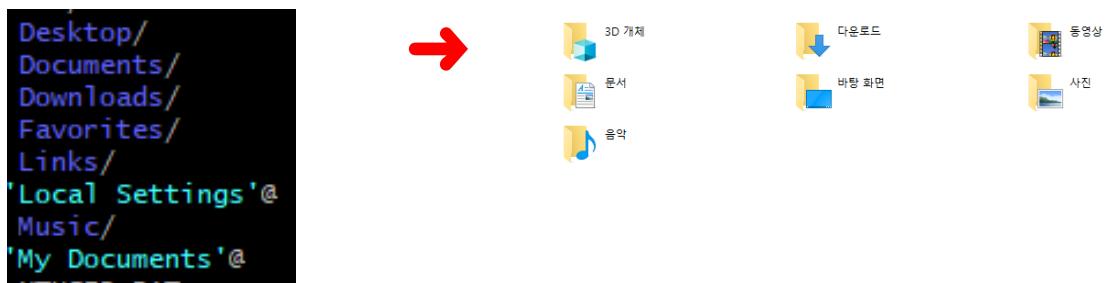
- 파일 위치 변경 정보 저장

## 유저프로그램(=응용프로그램)

- Widows 탐색기, 메모장, 카카오톡 ... 우리가 실행하는 모든 프로그램

## GUI

- 사용자가 편리하게 사용할 수 있도록 입출력 등의 기능을 알기 쉬운 **아이콘 따위의 그래픽으로 나타낸** 것



## GUI

**CLI(명령줄 인터페이스)**

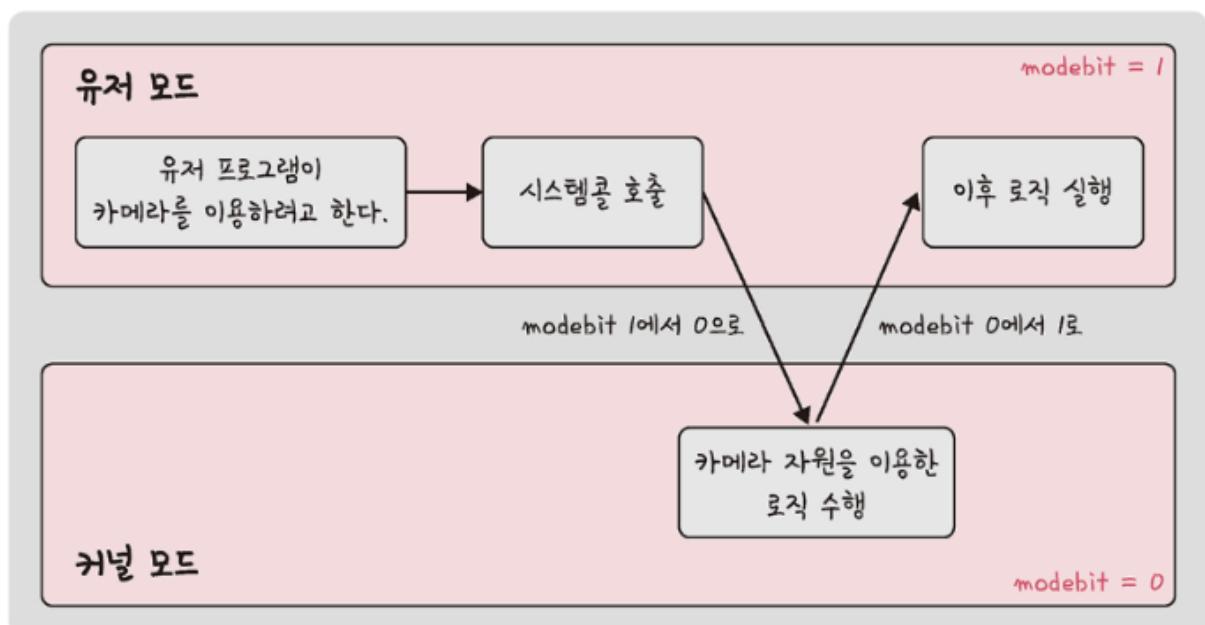
→ 글자를 입력하여 컴퓨터  
에 명령

`ls` → `cd Picture`

→ 마우스 클릭만으로도 컴퓨터와 상호작용 가능

## 시스템 콜

- 운영체제가 커널에 접근하기 위한 인터페이스
- 유저 프로그램이 운영체제의 서비스를 받기 위해 **커널 함수를 호출할 때 사용**
- `modebit`을 참고해서 유저 모드와 커널 모드를 구분(1, 0)
  - `modebit` → 1 또는 0의 값을 가지는 **플래그 변수**



**유저 모드** → 제한된 자원에만 접근 가능

**커널 모드** → 시스템 자원에 대한 접근 권한이 높음

## ? 유저 모드와 커널 모드를 구분하는 이유?

### 1. 보안

- 사용자 프로그램이 무분별하게 시스템 자원에 접근하는 것을 막을 수 있음

### 2. 보호 매커니즘

- 사용자의 잘못된 명령으로 시스템을 손상시키는 것을 방지

### 3. 안정성

- 유저 모드에서 실행되는 프로그램들은 **제한된 환경에서 동작**하므로, 프로그램의 오류 또는 충돌이 시스템 전체를 영향을 끼치지 않게 함

**유저 모드:** 유저 모드는 마치 은행 고객이 ATM을 사용하는 것과 비슷합니다. 고객은 ATM을 통해 계좌 조회, 입출금 등의 기본적인 작업을 수행할 수 있지만, 은행의 핵심 시스템에 직접적으로 접근하거나 변경할 수는 없습니다.

**커널 모드:** 커널 모드는 은행의 시스템 운영자에 해당할 수 있습니다. 시스템 운영자는 은행의 핵심 시스템과 데이터베이스에 접근하여 전반적인 시스템을 관리하고 보안을 유지합니다.

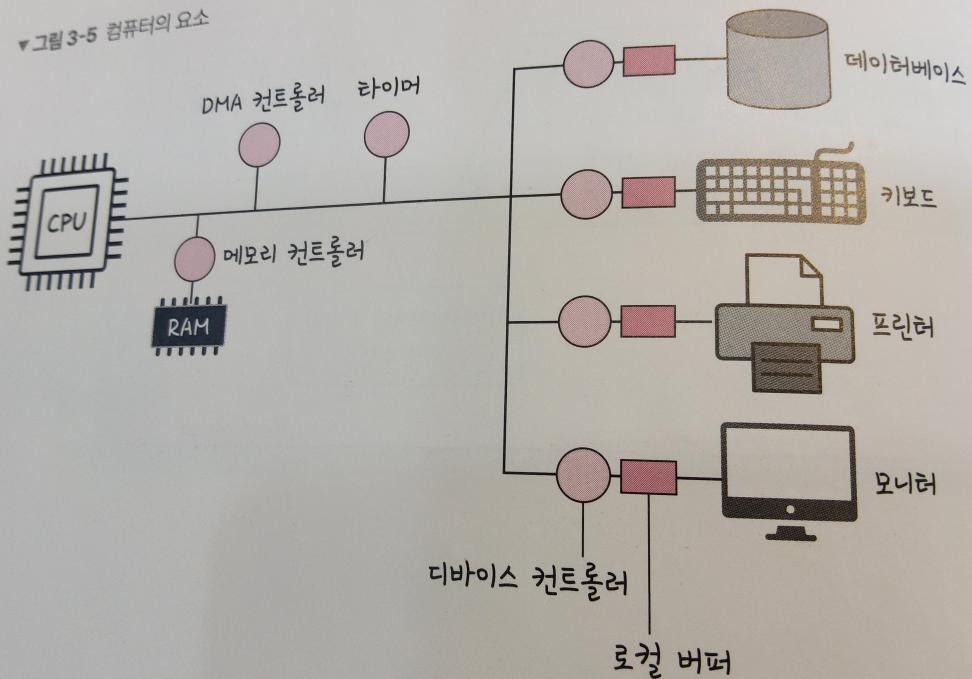
## 컴퓨터의 요소

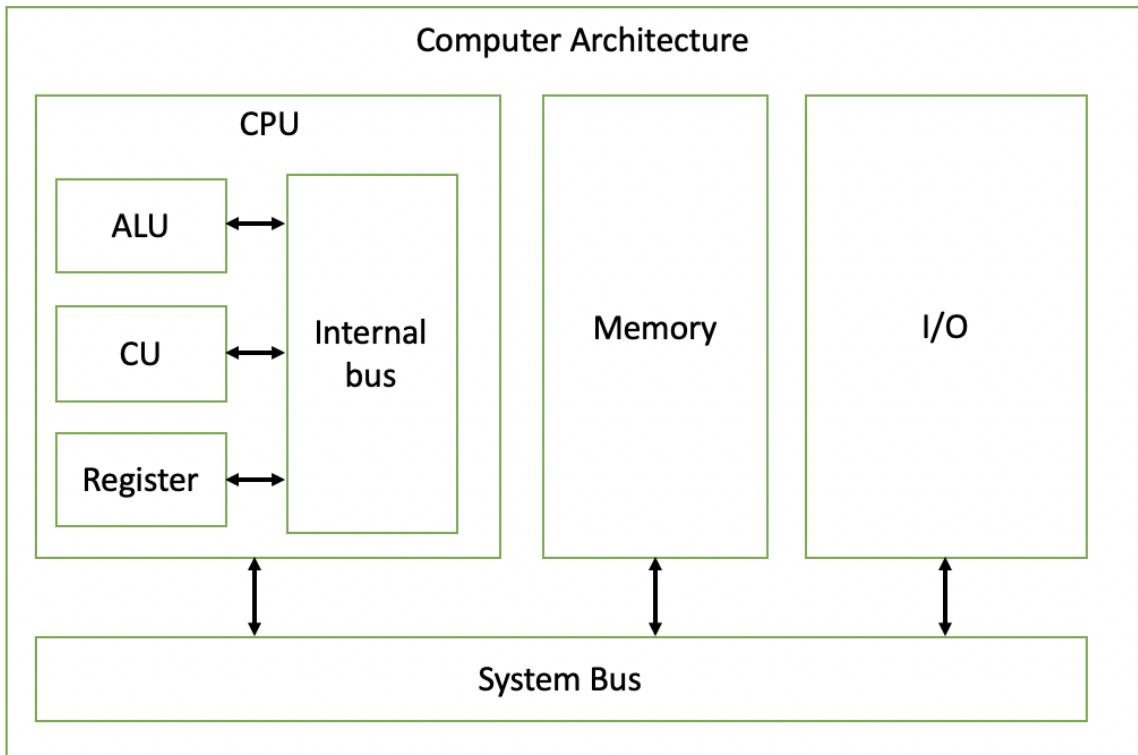
<책>

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

컴퓨터는 CPU, DMA  
합니다.

▼ 그림 3-5 컴퓨터의 요소





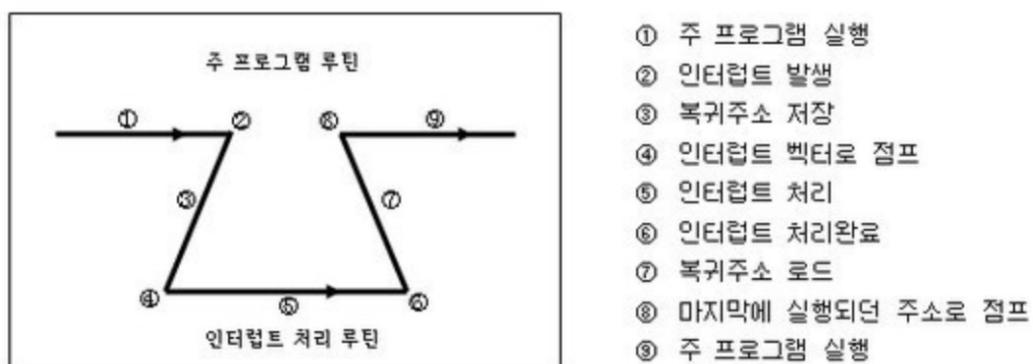
## CPU

- 산술논리장치(ALU), 제어장치(CU), 레지스터(Register)로 구성되어 있는 컴퓨터 장치
- 메모리에 존재하는 명령어를 해석
  - **산술논리장치(Arithmetic Logic Unit)**
    - 산술 연산, 논리 연산
  - **제어장치(Control Unit):**
    - 프로세스 조작 지시
    - 입출력장치 간 통신을 제어
    - 데이터 처리를 위한 순서를 결정
  - **레지스터**
    - CPU 안에 있는 매우 빠른 임시기억장치

- CPU와 직접 연결되어 있으므로 연산 속도가 메모리보다 수십 배에서 수백 배까지 빠름
- CPU는 자체적으로 데이터를 저장할 방법이 없기에 레지스터를 거쳐 데이터를 전달

## 인터럽트

- 어떤 신호가 들어왔을 때 **CPU를 잠깐 정지시킨** 뒤, 인터럽트 처리 루틴을 실행, 종료 후 다시 원래 작업으로 복귀하는 것
- 하드웨어 인터럽트, 소프트웨어 인터럽트
  - 하드웨어 인터럽트 - 키보드 연결이나 마우스 연결 등 I/O 디바이스에서 발생하는 인터럽트
  - 소프트웨어 인터럽트 - 프로세스 오류 등으로 프로세스가 시스템콜을 호출할 때 발생
- 인터럽트 핸들러 함수
  - 인터럽트가 발생했을 때 이를 핸들링하기 위한 함수. `request_irq()`를 통해 인터럽트 핸들러 함수를 등록 가능



## 폴링

- 대상을 주기적으로 감시하여 상황이 발생하면 해당 처리 루틴을 실행하여 처리

## ▼ 뇌피셜

### 1. 인터럽트:

- 손님이 호출벨을 누르면(인터럽트 발생), 시스템은 즉시 손님이 왔다는 사건에 반응하고, 주인은 손님을 확인하러 나가거나 문을 엽니다(인터럽트 처리)
- 이는 외부 사건(인터럽트)이 발생했을 때 시스템이 즉각적으로 대응하는 방식을 나타냅니다.

### 2. 폴링:

- 주인이 일정한 시간 간격으로 문을 열어보면서(폴링), 손님이 오지 않았는지 확인합니다. 정기적으로 주인이 확인하며 외부 사건을 감지하고 대응하는 방식을 폴링이라고 할 수 있습니다.



인터럽트는 외부 사건이 발생하면 바로 반응  
폴링은 주기적으로 확인하면서 상태를 감지하고 처리

## DMA 컨트롤러(Direct Memory Access)

- I/O 디바이스가 메모리에 직접 접근할 수 있도록 하는 하드웨어 장치이자 CPU의 일을 부담하는 보조 일꾼
- CPU에만 너무 많은 인터럽트 요청이 들어오기 때문에 인터럽트가 무수히 많아지면 CPU에 부하

## ▼ 뇌피셜

1. 일반적으로 메모리에 접근 하기 위해서는 CPU를 통해 접근해야함

2. CPU에만 너무 많은 인터럽트 요청이 들어오기 때문에 인터럽트가 무수히 많아  
지면 CPU에 부하가 오게 됨
3. DMA 컨트롤러를 이용하여 I/O 디바이스가 메모리에 직접 접근할 수 있게 하여  
CPU의 부담을 줄인다.

→ DMA 컨트롤러는 I/O 디바이스와 메모리 간의 데이터 전송을 효율적으로 관리하여 CPU의 부담을 줄이는 역할을 한다.

## Memory

- 데이터나 상태, 명령어 등을 기록하는 장치
- CPU는 계산을 담당하고, 메모리는 기억을 담당
- CPU는 일꾼이고, 메모리는 작업장이다. 작업장이 클수록 많은 일을 할 수 있듯이 메모리가 크면 클수록 많은 일을 동시에 할 수 있다.

## 타이머

- 특정 프로그램에 시간 제한을 다는 역할
- 시간이 많이 걸리는 프로그램이 작동할 때 제한을 걸기 위해서 존재

## 디바이스 컨트롤러

- 컴퓨터와 연결되어 있는 IO 디바이스들의 작은 CPU를 말한다.

### ▼ 뇌피셜 (쓸모 X)

**⚠️ 이러한 비유를 통해서 일부 개념을 이해하는 데 도움이 될 수 있습니다만, 몇 가지 정확하지 않거나 약간의 오해를 불러일으킬 수 있는 부분이 있습니다. ⚠️**

- CPU** - 옥수수 따는 노예 집합소
- 캐시 메모리** - 노예가 짊어지고 있는 백팩, 리어카까지 옥수수 안옮겨도 됨
- 램(메모리)** - 노예가 준비한 리어카  
램이 많을수록 좋은 이유 → 옥수수는 많이 땖는데 리어카가 작으면 한번에 창고로 못가져감
- 하드 디스크** - 수확한 옥수수 저장창고

출처 인터넷 커뮤니티

# VISUAL DIVE

## 컴알못을 위한 노예 비유법

코어의 수	쓰레드	오버클럭
네가 부릴 노예의 수	네가 부릴 노예의 손 갯수	노예가 발까지 써서 옥수수 수확

캐쉬 메모리	램	하드 디스크
노예가 짊어지고 있는 백팩	노예가 준비한 리어카	수확한 옥수수 저장창고

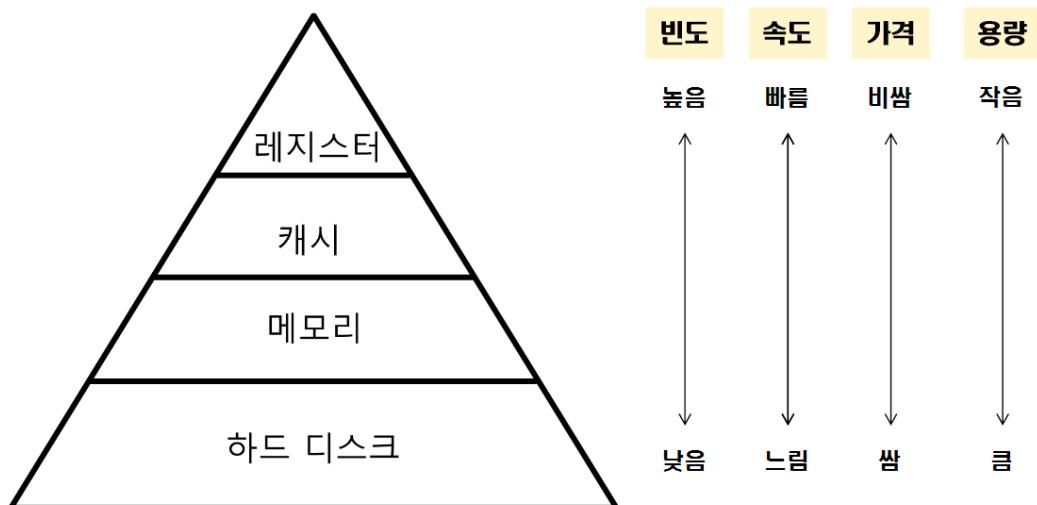
i3, i5, i7	i7 10900	램 다다익선 이유
노예가 3마리, 5마리, 7마리 (실제 코어 수와 다름. 이해를 돋기 위해 숫자 통일한 것)	7마리 10년도 생 노예의 손이 900개 (7마리의 10년에 태어난 노예들이 손이 900개라 수확도 빠름)	옥수수를 많이 따도 리어카가 작으면 한 번에 창고로 못 가져감

그래픽 카드 = gpu	파워	ssd	NVme ssd
옥수수 말고 감자 캐는 거 시킬 노예	노예에게 주는 월급 부족하면 의해해서 일 못함	최신식으로 좋은 저장창고	더 업그레이드해서 정말 빠르고 좋은 저장창고

기획 방현규 디자인 이영환

## 3.2 메모리

### 메모리 계층



### 캐시

- 데이터를 미리 복사해 놓는 임시 저장소이자 빠른 장치와 느린 장치에서 속도 차이에 따른 병목 현상을 줄이기 위한 메모리
- 메모리와 CPU 사이의 속도 차이가 너무 크기 때문에 이를 해결하기 위해 존재
- CPU와 메인 메모리 사이에 위치



## 병목현상이란?

시스템의 다른 구성 요소들이나 작업이 CPU의 처리 속도를 따라가지 못하기 때문에 CPU가 일부 시간 동안 대기 상태에 있게 됨, 이로 인해 CPU의 활용도가 낮아질 수 있으며, 전반적인 성능 저하가 발생할 수 있다.



### ⚠️ 노피설 ⚠️

교통체증은 도로에서 차량이 많아서 움직이기가 어려운 상황

마찬가지로, 컴퓨터에서도 특정한 작업들이 많아서 원활하게 진행되지 않는 상황이 있다.

예를 들어, 여러 프로그램이 동시에 컴퓨터의 '도로' 같은 부분을 사용하려고 할 때, 그 부분이 혼잡해지면 작업들이 원활하게 처리되지 않고 대기하게 됨

이 상황이 바로 병목 현상

## 지역성

- 캐시에는 '지역성'이라는 개념이 존재  
이러한 지역성은 캐시의 성능을 향상시키는 중요한 요소
- 시간 지역성, 공간 지역성

## 1. 시간 지역성

- 최근 사용한 데이터에 다시 접근하려는 특성
- 한번 접근한 데이터는 잠시 후에 다시 접근할 확률이 높다

```
arr = [0] * 10
print(arr)

"""
[ 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0 ]
"""

# 반복문을 통해 리스트의 원소에 접근하고 값 할당
for i in range(10):
    arr[i] = i

print(arr)

"""
[ 0, 1, 2, 3, 4,
  5, 6, 7, 8, 9 ]
"""


```

- 반복문으로 이루어진 코드 안의 변수 *i*에 계속해서 접근이 이루어짐
- 여기서 데이터는 변수 *i*이고 최근에 사용했기 때문에 계속 접근해서 +1을 연이어 하는 것을 볼 수 있다.

## 2. 공간 지역성

- 특정 데이터에 접근한 경우, 그 근처의 데이터에도 접근할 가능성이 높은 개념
- 리스트 원소에 대한 순차적인 접근

```
# 리스트를 생성하고 초기값으로 초기화
data_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```

# 반복문을 통해 리스트의 원소에 접근하고 값 출력
for item in data_list:
    print(item)

# 반복문 외부에서 다시 리스트의 원소에 접근하고 값 출력
for item in data_list:
    print(item)

```

- 첫 번째 반복문에서 리스트 'data\_list'의 각 원소에 대한 접근이 이루어진 이후, 두 번째 반복문에서는 반복문 외부에서 다시 리스트의 원소에 대한 접근이 이루어짐
- 첫 번째 반복문에서 이미 리스트의 원소에 대한 접근이 발생했으며, 이는 해당 데이터와 인접한 데이터가 캐시에 미리 로드될 가능성이 있다. → 공간 지역성

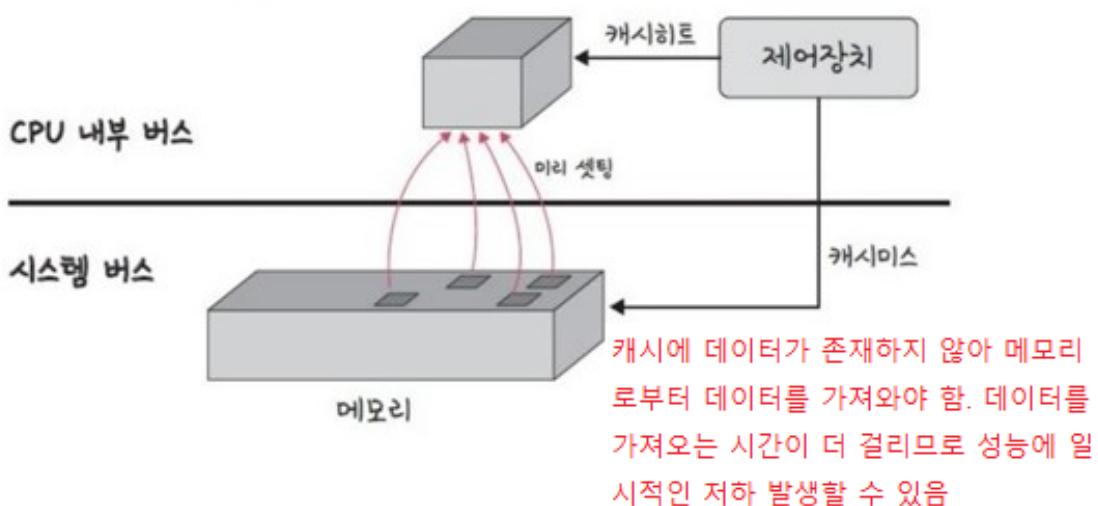
## 캐시히트

- CPU가 메모리에서 데이터를 요청했을 때, 해당 데이터가 **캐시에 이미 존재하는 경우**

## 캐시미스

- CPU가 메모리에서 데이터를 요청했을 때, 해당 데이터가 **캐시에 없어** 메모리로부터 가져와야 하는 경우

▼ 그림 3-9 캐시히트와 캐시미스



---

# 캐시 매팅

## 1. 직접 매팅

- 메모리 주소와 캐시의 순서를 일치시키는 방법
- 메모리가 1 ~ 100까지 있고 캐시가 1 ~ 10까지 있다면
  - 1~10까지의 메모리는 캐시의 1에 위치 / 11~20까지의 메모리는 캐시의 2에 위치
  - .....
- 구현이 매우 간단하지만, 적중률이 낮고 성능이 낮다
- 규칙을 만족시키면서 캐시를 넣다 보면 자주 교체되게 됨
  - ex) 메모리의 30~40에 해당하는 값을 자주 불러 사용해야 하는데 이를 저장하는 캐시 공간은 3으로 고정되어 있음

## 2. 연관매팅

- 순서를 일치시키지 않고 관련 있는 캐시와 메모리를 매팅함
- 충돌이 적지만 모든 블록을 탐색해야 해서 속도가 느리다.

## 3. 직접연관매팅

- 직접 매팅과 연관 매팅을 합쳐 놓은 방식
- 순서는 일치시키지만 집합을 둬서 저장하며 블록화되어 있기 때문에 효율적인 검색이 가능하다.
- 메모리가 1 ~ 100까지 있고 캐시가 1 ~ 10까지 있다면
  - 캐시 1~5에는 1~50의 데이터를 무작위로 저장

# 웹 브라우저의 캐시

## 쿠키

- 만료기한이 있는 키-값 저장소, 보통 서버에서 만료기한을 정함
- 사용자가 로그인하면 서버에서 생성한 세션 ID를 쿠키에 저장, 이를 통해 사용자의 세션을 식별

ex) 쇼핑 웹사이트에서 사용자가 장바구니에 넣은 정보들을 저장, 사용자가 페이지를 나갔다가 들어와도 장바구니 내의 상품이 보존되어 있음

## 로컬 스토리지

- 만료기한이 없는 키-값 저장소, HTML5를 지원하지 않는 웹 브라우저에서는 사용이 불가하며 클라이언트에서만 수정 가능함
  - 로컬 스토리지를 활용하여 웹 애플리케이션에서 오프라인 모드에서도 필요한 데이터를 저장
- ex) 사용자의 언어, 테마, 폰트 크기 등과 같은 환경 설정을 로컬 스토리지에 저장하여 사용자가 웹사이트를 사용할 때마다 같은 환경을 제공할 수 있다.

## 세션 스토리지

- 만료기한이 없는 키-값 저장소, HTML5를 지원하지 않는 웹 브라우저에서는 사용이 불가하며 클라이언트에서만 수정 가능함
  - 사용자가 브라우저를 닫을 때까지만 유효한 데이터를 저장할 수 있다.
- ex) 사용자가 로그인하면 세션 스토리지에 인증 토큰을 저장하여, 세션 동안 인증된 상태를 유지할 수 있다.

→ 이러한 것들은 보통 사용자의 커스텀한 정보나 인증 모듈 관련사항들을 웹 브라우저에 저장해서 추후 서버에 요청할 때 자신을 나타내는 아이덴티티나 중복 요청 방지를 위해 쓰인다.

## 가상 메모리

- 초창기 컴퓨터에서는 사용 가능한 RAM의 용량이 가장 큰 실행 애플리케이션의 주소 공간보다 커야 했음. 그렇지 않을 경우 ‘메모리 부족’ 오류로 실행이 불가하였다.
  - 애플리케이션을 실행하는 데 최소한 얼마만큼의 메모리가 필요한가?
  - 애플리케이션의 실행에 필요한 일부분만 메모리에 올려놓고 애플리케이션의 나머지 부분은 디스크에 남게 됨
- 메모리 관리 기법의 하나로 컴퓨터가 실제로 이용 가능한 메모리 자원을 추상화하여 이를 사용하는 사용자들에게 매우 큰 메모리로 보이게 만드는 것
- 각 프로그램에 실제 메모리 주소가 아닌 가상의 메모리 주소를 주는 방식
  - 가상적으로 주어진 주소를 가상 주소, 실제 메모리상에 있는 주소를 실제 주소라고 한다.
  - 메모리관리장치(MMU)에 의해 실제 주소로 변환되기에 사용자는 실제 주소를 의식 할 필요 없이 프로그램 구축 가능

### <핵심>

1. 메모리 관리 기법의 하나로 컴퓨터가 실제로 이용 가능한 메모리 자원을 추상화하여 이를 사용하는 사용자들에게 매우 큰 메모리로 보이게 하는것
2. 가상 주소 → MMU → 실제 주소
3. 용량이 작고 빠른 RAM + 용량이 크고 느린 디스크를 병합
  - 하나의 크고 빠른 기억장치(가상 메모리)처럼 동작하게 하는 것
4. ‘페이지 테이블’로 가상 주소와 실제 주소를 관리

## 5. 가상 메모리에는 존재하지만 실제 메모리인 RAM에는 현재 데이터가 없는데 접근하는 경우 → ‘페이지 폴트’발생

이때, 운영체제가 다음과 같은 과정을 통해 ‘페이지 폴트’가 발생하지 않은 것처럼 프로그램이 작동하게 해줌

1. 페이지 폴트가 발생하면 CPU는 이를 감지하고 운영체제에게 트랩(소프트웨어 인터럽트)을 발생시켜 예외 상황임을 알립니다.
2. 운영체제는 CPU의 동작을 잠시 멈춤
3. 페이지 테이블을 확인하여 가상 메모리에 페이지가 존재하는지 확인하고, 없으면 프로세스를 중단하고 현재 물리 메모리에 비어 있는 프레임이 있는지 찾는다. 물리 메모리에도 없다면 스와핑이 발생됨
4. 비어 있는 프레임에 해당 페이지를 로드하고, 페이지 테이블을 최신화
5. 중단되었던 CPU를 다시 시작한다.

## 6. 스레싱

- 시스템이 지나치게 많은 페이지 폴트로 인해 **대부분의 시간을 페이지 교체 작업에 사용하게 되어 유용한 작업을 거의 수행하지 못하는 상태**
- 메모리를 늘리거나, HDD를 사용한다면 SSD로 바꿔서 속도를 높인다.

## 7. 페이지 폴트 빈도를 줄이는 방법

### 작업 세트(working set)

- 프로세스의 과거 사용 이력인 지역성을 통해 결정된 페이지 집합을 만들어서 미리 메모리에 로드, 이를 통해 탐색에 드는 비용을 줄일 수 있음

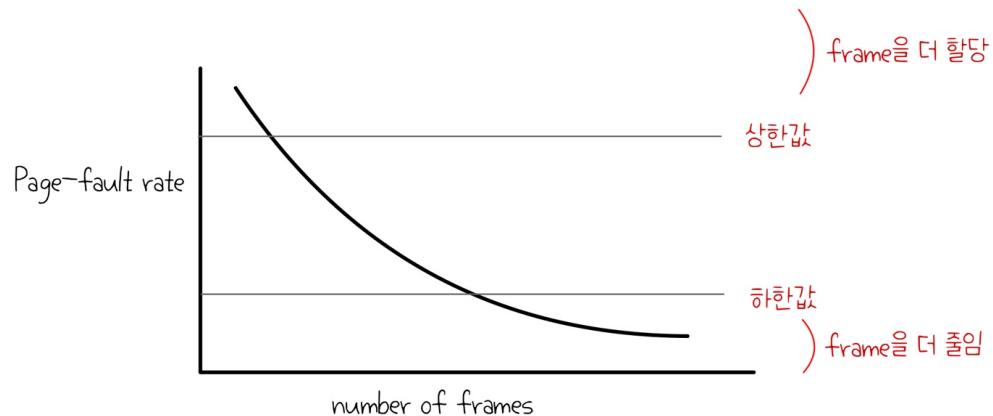
### 캐시와의 차이점?

둘 다 메모리의 사용 패턴을 파악하고 최적화하는 데에 관련이 있지만, 워킹 세트는 주로 운영체제에서 **프로세스의 메모리 관리**에 사용되며, 캐시는 하드웨어 수준에서 **빠른 데이터 액세스**를 위해 사용

## PFF(Page Fault Frequency)

- 페이지 폴트의 상한선과 하한선을 만드는 방법

- 상한선에 도달한다면 페이지를 늘리고 하한선에 도달한다면 페이지를 줄임



## 메모리 할당(연속 할당, 불연속 할당)

### 연속 할당

- 고정 분할 방식, 가변 분할 방식(최초적합, 최적적합, 최악적합)

#### 고정 분할 방식

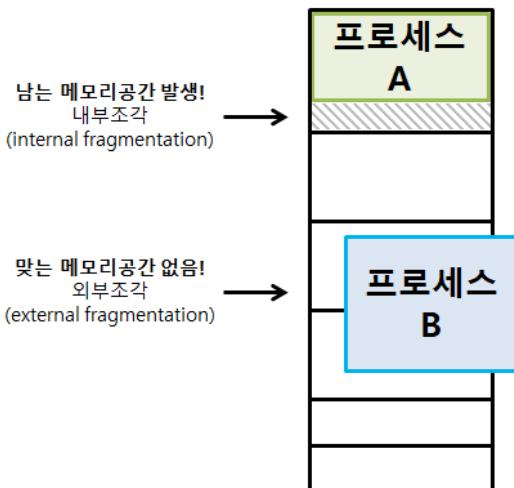
메모리의 크기를 미리 나누어 관리  
융통성이 없음, **내부 단편화** 발생

#### 가변 분할 방식

매 시점 프로그램의 크기에 맞게 동적으로 메모리를 나눠 사용(**최초적합**, **최적적합**, **최악적합**)



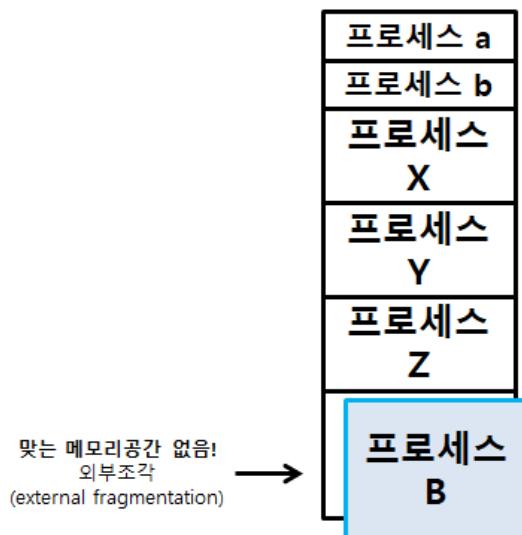
**최초적합** - 제일 먼저 맞는 곳에 넣음



내부 단편화: 프로그램의 크기가 메모리 공간보다 작아 다른 프로그램이 들어가지 못하는 공간이 많이 발생하는 현상

**최적적합** - 프로세스의 크기 이상인 공간 중 가장 작은 공간부터 할당

**최악적합** - 프로세스의 크기와 가장 많이 차이가 나는 공간에 할당



외부 단편화: 메모리를 나눈 크기보다 프로그램이 커서 들어가지 못하는 공간이 많이 발생하는 현상

## 불연속 할당

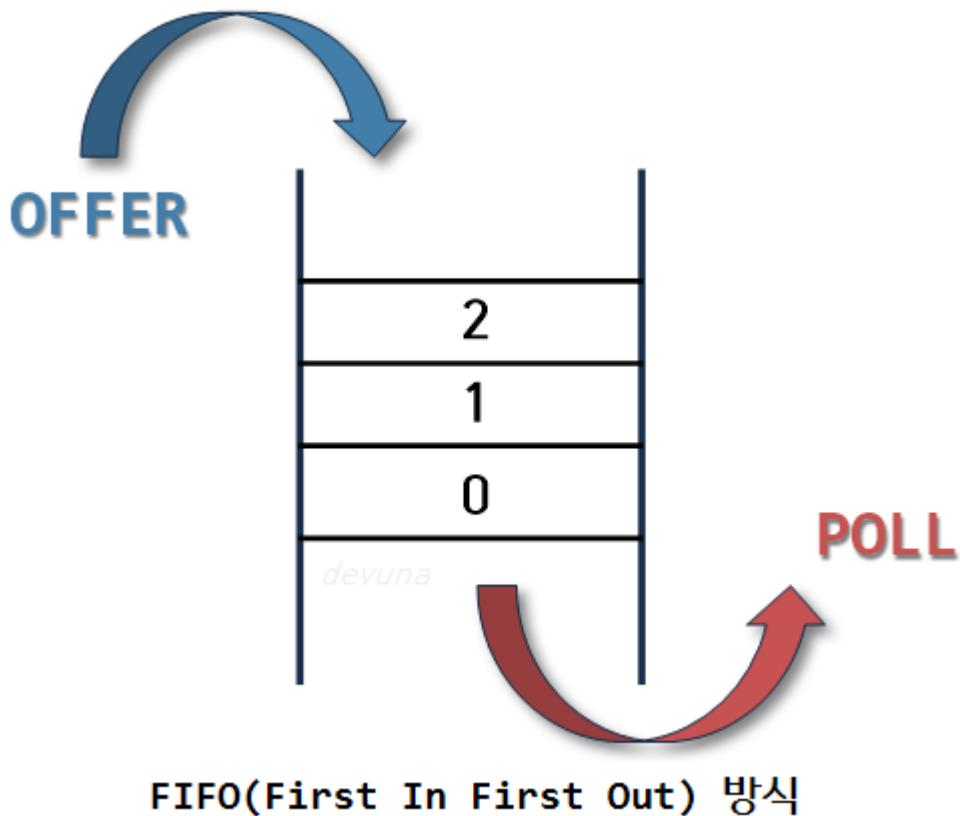
- 페이지, 세그멘테이션

## 페이지 교체 알고리즘

- FIFO, LRU, NRU, LFU

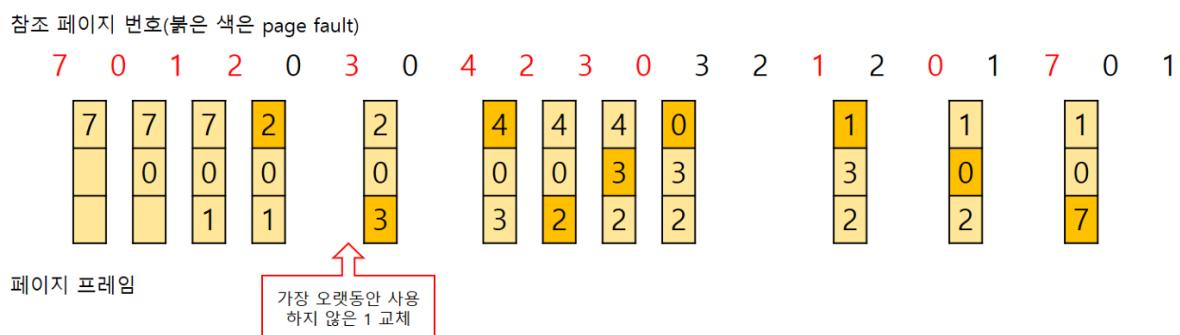
## FIFO(First In First Out)

- 가장 먼저 온 페이지를 가장 먼저 교체



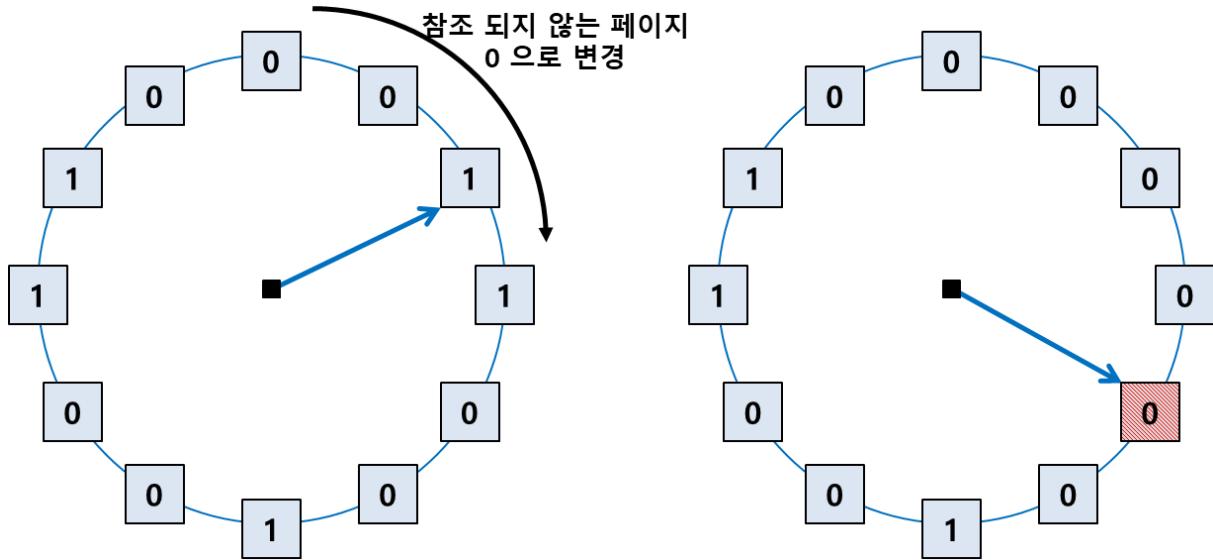
## LRU(Least Recently Used)

- 참조가 가장 오래된 페이지를 바꾼다.  
(페이지 참조 순서 1 → 2 → 3 → 4)



## NUR(Not Used Recently) - 이건 모르겠음

- LRU처럼 가장 최근에 참조되지 않은 페이지를 대상으로 선정
- 교체되는 페이지의 참조 시점이 가장 오래되었다는 것을 보장하지는 못한다.
- 클럭 알고리즘



## LFU(Least Frequently Used)

- 가장 참조 횟수가 적은 페이지를 교체 즉, 많이 사용되지 않은 것을 교체한다

요구 페이지 (demand page)	1	2	3	4	5	6	7	8	9	10	순서 페이지
	A	B	C	D	B	A	B	A	C	A	
old	A(1)	A(1)	A(1)	D(1)	D(1)	A(1)	A(1)	A(2)	A(2)	A(3)	
물리 메모리	Frame2	Frame3									
new											
상태	실패	실패	실패	실패	성공	실패	성공	성공	성공	성공	