# 2주차

| | |
|---|---|
| 진행 상태 | 시작 전 |
| 완료 확인 | ☐ |

## ▼ 03장 운영체제

### 3.1 운영체제와 컴퓨터

운영체제(OS) 는 사용자가 컴퓨터를 쉽게 다루게 해주는 인터페이스이며, 한정된 메모리나 시스템 자원을 효율적으로 분배하는 역할

#### 3.1.1 운영체제의 역할과 구조

**운영체제의 역할 (4가지)**

1. **CPU 스케줄링과 프로세스 관리**: CPU 소유권을 어떤 프로세스에 할당할지, 프로세스의 생성과 삭제, 자원 할당 및 반환을 관리
2. **메모리 관리**: 한정된 메모리를 어떤 프로세스에 얼만큼 할당해야 하는지 관리
3. **디스크 파일 관리**: 디스크 파일을 어떠한 방법으로 보관할지 관리
4. **I/O 디바이스 관리**: I/O 디바이스들인 마우스, 키보드와 컴퓨터 간에 데이터를 주고받는 것을 관리

**운영체제의 구조**

- **상위부터**: 유저 프로그램 → GUI → 시스템콜 → 커널 → 드라이버 → 하드웨어
- **운영체제 구성요소**: GUI, 시스템콜, 커널, 드라이버

**시스템콜**

- **정의**: 운영체제가 커널에 접근하기 위한 인터페이스
- **역할**: 유저 프로그램이 운영체제의 서비스를 받기 위해 커널 함수를 호출할 때 사용
- **과정**: 유저 프로그램이 I/O 요청으로 트랩(trap) 발동 → 올바른 I/O 요청인지 확인 → 유저 모드에서 시스템콜을 통해 커널 모드로 변환하여 실행

**modebit**

- **역할**: 시스템콜이 작동될 때 유저 모드와 커널 모드를 구분하는 플래그 변수
- **값**: 0 = 커널 모드, 1 = 유저 모드
- **과정**: 유저 프로그램이 자원 이용 시 → 시스템콜 호출 → modebit를 1에서 0으로 변경(커널 모드) → 자원 이용 로직 수행 → modebit를 0에서 1로 변경(유저 모드)

#### 3.1.2 컴퓨터의 요소

컴퓨터는 **CPU, DMA 컨트롤러, 메모리, 타이머, 디바이스 컨트롤러** 등으로 구성

**CPU (Central Processing Unit)**

- **구성**: 산술논리연산장치, 제어장치, 레지스터
- **역할**: 인터럽트에 의해 메모리에 존재하는 명령어를 해석하여 실행하는 일꾼

**제어장치 (CU, Control Unit)**

- 프로세스 조작을 지시하는 CPU의 한 부품
- 입출력장치 간 통신 제어, 명령어 읽기/해석, 데이터 처리 순서 결정

**레지스터**

- CPU 안에 있는 매우 빠른 임시기억장치
- CPU와 직접 연결되어 메모리보다 수십 배~수백 배 빠름
- CPU는 자체적으로 데이터 저장 불가능하므로 레지스터를 거쳐 데이터 전달

### 산술논리연산장치 (ALU, Arithmetic Logic Unit)

- 덧셈, 뺄셈 같은 산술 연산과 배타적 논리합, 논리곱 같은 논리 연산을 계산하는 디지털 회로

### CPU의 연산 처리 과정

1. **제어장치**가 메모리에 계산할 값을 로드하고 레지스터에도 로드
2. **제어장치**가 레지스터에 있는 값을 계산하라고 산술논리연산장치에 명령
3. **제어장치**가 계산된 값을 레지스터에서 메모리로 저장

### 인터럽트

- **정의**: 어떤 신호가 들어왔을 때 CPU를 잠깐 정지시키는 것
- **발생 원인**: 키보드/마우스 등 IO 디바이스, 0으로 숫자를 나누는 산술 연산, 프로세스 오류 등
- **처리**: 인터럽트 핸들러 함수가 모여 있는 인터럽트 벡터로 가서 인터럽트 핸들러 함수 실행

### 하드웨어 인터럽트

- 키보드 연결, 마우스 연결 등 IO 디바이스에서 발생하는 인터럽트
- 인터럽트 라인 설계 후 순차적 인터럽트 실행 중지 → 운영체제에 시스템콜 요청 → 디바이스의 로컬 버퍼에 접근하여 작업 수행

### 소프트웨어 인터럽트

- 트랩(trap) 이라고도 함
- 프로세스 오류 등으로 프로세스가 시스템콜을 호출할 때 발동

### DMA 컨트롤러

- **역할**: I/O 디바이스가 메모리에 직접 접근할 수 있도록 하는 하드웨어 장치
- **목적**: CPU에 너무 많은 인터럽트 요청이 들어오는 것을 방지하여 CPU 부하를 막아주는 보조 일꾼
- **기능**: CPU와 DMA 컨트롤러가 동시에 같은 작업을 하는 것을 방지

### 메모리

- **정의**: 전자회로에서 데이터나 상태, 명령어 등을 기록하는 장치 (보통 RAM을 지칭)
- **역할**: CPU는 계산 담당, 메모리는 기억 담당
- **비유**: 공장에서 CPU는 일꾼, 메모리는 작업장 (작업장이 클수록 많은 일을 동시에 처리 가능)

### 타이머

- **역할**: 특정 프로그램에 시간 제한을 다는 역할
- **목적**: 시간이 많이 걸리는 프로그램이 작동할 때 제한을 걸기 위해 존재

### 디바이스 컨트롤러

- **정의**: 컴퓨터와 연결되어 있는 IO 디바이스들의 작은 CPU

---

# 3.2 메모리

CPU는 메모리에 올라와 있는 프로그램의 명령어들을 실행

## 3.2.1 메모리 계층

### 메모리 계층 구조 (위에서부터)

1. **레지스터**: CPU 안에 있는 작은 메모리, 휘발성, 속도 가장 빠름, 기억 용량 가장 적음
2. **캐시**: L1, L2 캐시, 휘발성, 속도 빠름, 기억 용량 적음
3. **주기억장치**: RAM, 휘발성, 속도 보통, 기억 용량 보통
4. **보조기억장치**: HDD, SSD, 비휘발성, 속도 낮음, 기억 용량 많음

### 캐시

- **정의**: 데이터를 미리 복사해 놓는 임시 저장소이자 빠른 장치와 느린 장치 간 속도 차이에 따른 병목 현상을 줄이기 위한 메모리
- **목적**: 데이터 접근 시간이 오래 걸리는 경우 해결, 재계산 시간 절약

### 지역성의 원리

- **시간 지역성**: 최근 사용한 데이터에 다시 접근하려는 특성
- **공간 지역성**: 최근 접근한 데이터를 이루고 있는 공간이나 그 가까운 공간에 접근하는 특성

### 캐시히트와 캐시미스

- **캐시히트**: 캐시에서 원하는 데이터를 찾은 경우 (빠름)
- **캐시미스**: 해당 데이터가 캐시에 없어서 주 메모리로 가서 데이터를 찾아오는 경우 (느림)

### 캐시매핑

- **직접 매핑**: 처리 빠르지만 충돌 발생 잦음
- **연관 매핑**: 순서 일치시키지 않고 관련 있는 캐시와 메모리 매핑, 충돌 적지만 모든 블록 탐색으로 속도 느림
- **집합 연관 매핑**: 직접 매핑과 연관 매핑을 합친 방식, 순서는 일치시키지만 집합을 둬서 저장

### 웹 브라우저의 캐시

- **쿠키**: 만료기한이 있는 키-값 저장소, 4KB까지 저장, 다른 도메인에서 자동 전송
- **로컬 스토리지**: 만료기한이 없는 키-값 저장소, 10MB까지 저장, 브라우저 닫아도 유지
- **세션 스토리지**: 만료기한이 없는 키-값 저장소, 5MB까지 저장, 탭 단위로 생성, 탭 닫으면 삭제

## 3.2.2 메모리 관리

### 가상 메모리

- **정의**: 컴퓨터가 실제로 이용 가능한 메모리 자원을 추상화하여 사용자들에게 매우 큰 메모리로 보이게 만드는 것
- **가상 주소**: 가상적으로 주어진 주소 (logical address)
- **실제 주소**: 실제 메모리상에 있는 주소 (physical address)
- **변환**: 메모리관리장치(MMU)에 의해 가상 주소가 실제 주소로 변환
- **관리**: 페이지 테이블로 관리, 속도 향상을 위해 TLB 사용

### 스와핑

- **정의**: 가상 메모리에는 존재하지만 실제 메모리인 RAM에는 현재 없는 데이터나 코드에 접근할 경우 발생
- **과정**: 메모리에서 당장 사용하지 않는 영역을 하드디스크로 옮기고 하드디스크의 일부분을 마치 메모리처럼 불러와 사용

### 페이지 폴트

- **정의**: 프로세스의 주소 공간에는 존재하지만 지금 컴퓨터의 RAM에는 없는 데이터에 접근했을 경우 발생
- **처리 과정**:
  1. CPU가 물리 메모리 확인하여 해당 페이지가 없으면 트랩 발생해서 운영체제에 알림
  2. 운영체제가 CPU 동작을 잠시 멈춤
  3. 운영체제가 페이지 테이블 확인하여 가상 메모리에 페이지 존재하는지 확인, 없으면 프로세스 중단하고 물리 메모리에 비어 있는 프레임 찾음 (없다면 스와핑 발동)
  4. 비어 있는 프레임에 해당 페이지 로드하고 페이지 테이블 최신화
  5. 중단되었던 CPU를 다시 시작

### 스레싱

- **정의**: 메모리의 페이지 폴트율이 높은 것, 컴퓨터의 심각한 성능 저하 초래
- **원인**: 메모리에 너무 많은 프로세스가 동시에 올라가게 되면 스와핑이 많이 일어나서 발생

- **해결방법**:
  - 메모리를 늘리거나 HDD를 SSD로 교체
  - **작업 세트**: 프로세스의 과거 사용 이력인 지역성을 통해 결정된 페이지 집합을 미리 메모리에 로드
  - **PFF**: 페이지 폴트 빈도를 조절하는 방법으로 상한선과 하한선을 만드는 방법

### 메모리 할당

### 연속 할당

- **고정 분할 방식**: 메모리를 미리 나누어 관리, 융통성 없음, 내부 단편화 발생
  - **가변 분할 방식**: 매 시점 프로그램 크기에 맞게 동적으로 메모리 분할, 내부 단편화 발생하지 않고 외부 단편화 발생 가능
    - **최초적합**: 위쪽이나 아래쪽부터 시작해서 홀을 찾으면 바로 할당
    - **최적적합**: 프로세스 크기 이상인 공간 중 가장 작은 홀부터 할당
    - **최악적합**: 프로세스 크기와 가장 많이 차이가 나는 홀에 할당

### 불연속 할당

- **페이징**: 동일한 크기의 페이지 단위로 나누어 메모리의 서로 다른 위치에 프로세스 할당
- **세그멘테이션**: 페이지 단위가 아닌 의미 단위인 세그먼트로 나누는 방식
- **페이지드 세그멘테이션**: 공유나 보안을 의미 단위의 세그먼트로 나누고, 물리적 메모리는 페이지로 나누는 것

### 페이지 교체 알고리즘

- **오프라인 알고리즘**: 먼 미래에 참조되는 페이지와 현재 할당하는 페이지를 바꾸는 알고리즘 (이론적으로 가장 좋지만 사용 불가)
- **FIFO**: 가장 먼저 온 페이지를 교체 영역에 가장 먼저 놓는 방법
- **LRU(**Least ***Recently*** Used 가장 오랫동안 사용되지 않은 페이지 교체) : 참조가 가장 오래된 페이지를 교체 (해시 테이블과 이중 연결 리스트로 구현)
- **NUR(Not Used Recently  최근에 사용하지 않은 페이지 교체)**: LRU에서 발전한 clock 알고리즘, 0과 1 비트를 두어 시계 방향으로 돌면서 0을 찾아 교체
- **LFU(**Least ***Frequently*** Used  참조 횟수가 가장 작은 페이지 교체): 가장 참조 횟수가 적은 페이지를 교체

---

# 3.3 프로세스와 스레드

## 3.3.1 프로세스와 컴파일 과정

### 프로세스

- **정의**: 컴퓨터에서 실행되고 있는 프로그램, CPU 스케줄링의 대상이 되는 작업
- **관계**: 프로그램이 메모리에 올라가면 프로세스가 되는 인스턴스화 발생

### 컴파일 과정

1. **전처리**: 소스 코드의 주석 제거, #include 등 헤더 파일 병합, 매크로 치환
2. **컴파일러**: 오류 처리, 코드 최적화 작업, 어셈블리어로 변환
3. **어셈블러**: 어셈블리어를 목적 코드(object code)로 변환 (.o 파일)
4. **링커**: 라이브러리 함수 또는 다른 파일들과 목적 코드를 결합하여 실행 파일 생성

### 라이브러리

- **정적 라이브러리**: 프로그램 빌드 시 라이브러리가 제공하는 모든 코드를 실행 파일에 삽입, 외부 의존도 낮음, 메모리 효율성 떨어짐
- **동적 라이브러리**: 프로그램 실행 시 필요할 때만 DLL 함수 정보를 통해 참조, 메모리 효율성 좋음, 외부 의존도 높음

## 3.3.2 프로세스의 상태

- **생성 상태**: 프로세스가 생성된 상태, fork() 또는 exec() 함수를 통해 생성, PCB 할당

- **대기 상태**: 메모리 공간이 충분하면 메모리 할당받고 CPU 스케줄러로부터 CPU 소유권이 넘어오기를 기다리는 상태
- **대기 중단 상태**: 메모리 부족으로 일시 중단된 상태
- **실행 상태**: CPU 소유권과 메모리를 할당받고 인스트럭션을 수행 중인 상태 (CPU burst)
- **중단 상태**: 어떤 이벤트가 발생한 이후 기다리며 프로세스가 차단된 상태
- **일시 중단 상태**: 중단된 상태에서 프로세스가 실행되려고 했지만 메모리 부족으로 일시 중단된 상태
- **종료 상태**: 메모리와 CPU 소유권을 모두 놓고 가는 상태

### fork()와 exec()

- **fork()**: 부모 프로세스의 주소 공간을 그대로 복사하여 새로운 자식 프로세스 생성
- **exec()**: 새롭게 프로세스를 생성하는 함수

## 3.3.3 프로세스의 메모리 구조

### 메모리 구조 (위에서부터)

1. **스택**: 지역변수, 매개변수, 함수 저장, 컴파일 시 크기 결정, 동적 특징, 위 주소부터 할당
2. **힙**: 동적 할당할 때 사용, 런타임 시 크기 결정, 동적 특징, 아래 주소부터 할당
3. **데이터 영역**: 전역변수, 정적변수 저장, 정적 특징
    - **BSS 영역**: 초기화되지 않은 변수가 0으로 초기화되어 저장
    - **Data 영역**: 0이 아닌 다른 값으로 할당된 변수들 저장
4. **코드 영역**: 프로그램에 내장되어 있는 소스 코드, 수정 불가능한 기계어로 저장, 정적 특징

## 3.3.4 PCB (Process Control Block)

### PCB 구조

- **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

- **정의**: PCB를 교환하는 과정
- **발생 원인**: 한 프로세스에 할당된 시간이 끝나거나 인터럽트에 의해 발생
- **과정**: 프로세스 A 실행 중단 → 프로세스 A의 PCB 저장 → 프로세스 B 로드하여 실행 → 프로세스 B의 PCB 저장 → 프로세스 A의 PCB 로드
- **비용**: 유휴 시간 발생, 캐시미스 발생 (프로세스가 가지고 있는 메모리 주소로 인한 캐시클리어 과정)

## 3.3.5 멀티프로세싱

### 특징

- **정의**: 여러 개의 프로세스를 통해 동시에 두 가지 이상의 일을 수행하는 것
- **장점**: 하나 이상의 일을 병렬 처리 가능, 특정 프로세스에 문제 발생해도 다른 프로세스로 처리 가능하여 신뢰성 높음

### 웹 브라우저의 멀티프로세스 구조

- **브라우저 프로세스**: 주소 표시줄, 북마크 막대, 뒤로/앞으로 가기 버튼 등 담당, 네트워크 요청이나 파일 접근 같은 권한 담당

- **렌더러 프로세스**: 웹 사이트가 '보이는' 부분의 모든 것을 제어
- **플러그인 프로세스**: 웹 사이트에서 사용하는 플러그인 제어
- **GPU 프로세스**: GPU를 이용해서 화면을 그리는 부분 제어

### IPC (Inter Process Communication)

- **정의**: 프로세스끼리 데이터를 주고받고 공유 데이터를 관리하는 메커니즘

### IPC 종류

1. **공유 메모리**: 여러 프로세스에 동일한 메모리 블록에 대한 접근 권한 부여, 가장 빠름, 동기화 필요
2. **파일**: 디스크에 저장된 데이터 또는 파일 서버에서 제공한 데이터 기반으로 통신
3. **소켓**: 동일한 컴퓨터의 다른 프로세스나 네트워크의 다른 컴퓨터로 네트워크 인터페이스를 통해 전송하는 데이터 (TCP, UDP)
4. **익명 파이프**: 프로세스 간에 FIFO 방식으로 읽히는 임시 공간인 파이프 기반, 단방향 방식, 부모-자식 프로세스 간에만 사용 가능
5. **명명된 파이프**: 파이프 서버와 하나 이상의 파이프 클라이언트 간의 통신을 위한 명명된 단방향 또는 이중 파이프, 다른 네트워크상의 컴퓨터와도 통신 가능
6. **메시지 큐**: 메시지를 큐 데이터 구조 형태로 관리, 커널에서 전역적으로 관리, 사용 방법이 직관적이고 간단

## 3.3.6 스레드와 멀티스레딩

### 스레드

- **정의**: 프로세스의 실행 가능한 가장 작은 단위
- **특징**: 코드, 데이터, 힙은 스레드끼리 서로 공유, 스택 영역은 각각 생성

### 멀티스레딩

- **정의**: 프로세스 내 작업을 여러 개의 스레드로 처리하는 기법
- **장점**:
  - 스레드끼리 서로 자원을 공유하기 때문에 효율성 높음
  - 한 스레드가 중단되어도 다른 스레드는 실행 상태일 수 있어 빠른 처리 가능
  - 동시성에 큰 장점
- **단점**: 한 스레드에 문제가 생기면 다른 스레드에도 영향을 끼쳐 프로세스 전체에 영향 줄 수 있음

## 3.3.7 공유 자원과 임계 영역

### 공유 자원

- **정의**: 시스템 안에서 각 프로세스, 스레드가 함께 접근할 수 있는 모니터, 프린터, 메모리, 파일, 데이터 등의 자원이나 변수
- **경쟁 상태**: 두 개 이상의 프로세스가 공유 자원을 동시에 읽거나 쓰는 상황

### 임계 영역

- **정의**: 둘 이상의 프로세스, 스레드가 공유 자원에 접근할 때 순서 등의 이유로 결과가 달라지는 코드 영역
- **해결 조건**: 상호 배제, 한정 대기, 융통성
- **해결 방법**: 뮤텍스, 세마포어, 모니터 (모두 잠금 메커니즘 기반)

### 뮤텍스

- **정의**: 프로세스나 스레드가 공유 자원을 lock()을 통해 잠금 설정하고 사용 후 unlock()을 통해 잠금 해제하는 객체
- **특징**: 잠금 또는 잠금 해제라는 상태만 가짐

### 세마포어

- **정의**: 일반화된 뮤텍스, 간단한 정수 값과 wait() 및 signal() 함수로 공유 자원에 대한 접근 처리
- **바이너리 세마포어**: 0과 1의 두 가지 값만 가질 수 있는 세마포어
- **카운팅 세마포어**: 여러 개의 값을 가질 수 있는 세마포어, 여러 자원에 대한 접근 제어에 사용

- **차이점**: 뮤텍스는 잠금 기반의 상호배제가 일어나는 '잠금 메커니즘', 세마포어는 신호 기반의 상호배제가 일어나는 '신호 메커니즘'

### 모니터

- **정의**: 둘 이상의 스레드나 프로세스가 공유 자원에 안전하게 접근할 수 있도록 공유 자원을 숨기고 해당 접근에 대해 인터페이스만 제공
- **특징**: 모니터큐를 통해 공유 자원에 대한 작업들을 순차적으로 처리
- **장점**: 세마포어보다 구현하기 쉬움, 상호 배제가 자동 (세마포어는 명시적으로 구현해야 함)

## 3.3.8 교착 상태

### 교착 상태 (Deadlock)

- **정의**: 두 개 이상의 프로세스들이 서로가 가진 자원을 기다리며 중단된 상태

### 교착 상태의 원인

1. **상호 배제**: 한 프로세스가 자원을 독점하고 있으며 다른 프로세스들은 접근 불가
2. **점유 대기**: 특정 프로세스가 점유한 자원을 다른 프로세스가 요청하는 상태
3. **비선점**: 다른 프로세스의 자원을 강제적으로 가져올 수 없음
4. **환형 대기**: 프로세스 A는 프로세스 B의 자원을 요구하고, 프로세스 B는 프로세스 A의 자원을 요구하는 등 서로가 서로의 자원을 요구하는 상황

### 교착 상태의 해결 방법

1. **예방**: 자원을 할당할 때 애초에 조건이 성립되지 않도록 설계
2. **회피**: 교착 상태 가능성이 없을 때만 자원 할당, **은행원 알고리즘** 사용 (프로세스당 요청할 자원들의 최대치를 통해 자원 할당 가능 여부 파악)
3. **탐지**: 교착 상태가 발생하면 사이클이 있는지 찾아보고 이에 관련된 프로세스를 한 개씩 제거
4. **회복**: 교착 상태는 매우 드물게 일어나기 때문에 이를 처리하는 비용이 더 커서 교착 상태가 발생하면 사용자가 작업을 종료 (현대 운영체제에서 채택)

---

# 3.4 CPU 스케줄링 알고리즘

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

### 목표

- CPU 이용률은 높게
- 주어진 시간에 많은 일을 하게
- 준비 큐(ready queue)에 있는 프로세스는 적게
- 응답 시간은 짧게 설정

## 3.4.1 비선점형 방식

### 특징

- 프로세스가 스스로 CPU 소유권을 포기하는 방식
- 강제로 프로세스를 중지하지 않음
- 컨텍스트 스위칭으로 인한 부하가 적음

### FCFS (First Come, First Served)

- **정의**: 가장 먼저 온 것을 가장 먼저 처리하는 알고리즘
- **단점**: 길게 수행되는 프로세스 때문에 '준비 큐에서 오래 기다리는 현상' 발생

### SJF (Shortest Job First)

- **정의**: 실행 시간이 가장 짧은 프로세스를 가장 먼저 실행하는 알고리즘

- **장점**: 평균 대기 시간이 가장 짧음
- **단점**: 긴 시간을 가진 프로세스가 실행되지 않는 현상 발생, 실제로는 실행 시간을 알 수 없기 때문에 과거의 실행 시간을 토대로 추측해서 사용

### 우선순위

- **정의**: SJF 스케줄링에서 긴 시간을 가진 프로세스가 실행되지 않는 현상을 오래된 작업일수록 우선순위를 높이는 방법(aging)을 통해 단점을 보완한 알고리즘

## 3.4.2 선점형 방식

### 특징

- 현대 운영체제가 쓰는 방식
- 지금 사용하고 있는 프로세스를 알고리즘에 의해 중단 시켜 버리고 강제로 다른 프로세스에 CPU 소유권을 할당하는 방식

### 라운드 로빈 (RR, Round Robin)

- **정의**: 현대 컴퓨터가 쓰는 스케줄링인 우선순위 스케줄링의 일종으로 각 프로세스는 동일한 할당 시간을 주고 그 시간 안에 끝나지 않으면 다시 준비 큐의 뒤로 가는 알고리즘
- **특징**:
  - q만큼의 할당 시간이 부여되고 N개의 프로세스가 운영된다면 (N-1) × q 시간이 지나면 자기 차례가 옴
  - 할당 시간이 너무 크면 FCFS가 되고 짧으면 컨텍스트 스위칭이 잦아져서 오버헤드(비용) 증가
  - 전체 작업 시간은 길어지지만 평균 응답 시간은 짧아짐
- **활용**: 로드밸런서에서 트래픽 분산 알고리즘으로도 사용

### SRF (Shortest Remaining Time First)

- **정의**: SJF는 중간에 실행 시간이 더 짧은 작업이 들어와도 기존 짧은 작업을 모두 수행하고 그다음 짧은 작업을 이어나가는데, SRF는 중간에 더 짧은 작업이 들어오면 수행하던 프로세스를 중지하고 해당 프로세스를 수행하는 알고리즘

### 다단계 큐

- **정의**: 우선순위에 따른 준비 큐를 여러 개 사용하고, 큐마다 라운드 로빈이나 FCFS 등 다른 스케줄링 알고리즘을 적용한 것
- **특징**:
  - 큐 간의 프로세스 이동이 안 되므로 스케줄링 부담이 적음
  - 유연성이 떨어지는 단점이 있음

**▼ 예상질문**

▼ 프로세스와 스레드의 차이점을 설명해주세요.

**프로세스**는 실행 중인 프로그램으로 운영체제로부터 독립적인 메모리 공간을 할당받습니다. **스레드**는 프로세스 내에서 실행되는 작업의 단위로, 같은 프로세스 내의 스레드들은 코드, 데이터, 힙 영역을 공유하고 스택 영역만 독립적으로 가집니다.

▼ 데드락(교착상태)이 발생하는 조건과 해결 방법을 설명해주세요.

**데드락**은 두 개 이상의 프로세스가 서로의 자원을 기다리며 무한정 대기하는 상황입니다.
발생 조건은 4가지가 있는데 상호배제/점유대기/비선점/환형대기가 있고, 해결 방법으로는

예방/회피/탐지 및 회복/무시가 있습니다.

▼ 가상 메모리와 페이지 교체 알고리즘에 대해 설명해주세요.

**가상 메모리**는 물리적 메모리의 한계를 극복하기 위해 디스크 공간을 메모리처럼 사용할 수 있게 하는 기술입니다. 프로세스는 가상 주소를 사용하고, MMU(Memory Management Unit)가 이를 물리 주소로 변환합니다.

**페이지 폴트**가 발생했을 때 물리 메모리가 가득 찬 상황에서는 페이지 교체 알고리즘을 사용합니다

**주요 페이지 교체 알고리즘:**

- **FIFO(First In First Out)**: 가장 먼저 들어온 페이지를 교체, 구현이 간단하지만 성능이 좋지 않음
- **LRU(Least Recently Used)**: 가장 오랫동안 사용되지 않은 페이지를 교체, 지역성 원리 활용으로 성능 우수

- **LFU(Least Frequently Used)**: 사용 빈도가 가장 낮은 페이지를 교체

# ▼ 04장 데이터베이스

## 4.1 데이터베이스의 기본

### 4.1.1 엔터티

- **엔터티**: 여러 속성을 지닌 명사 (사람, 장소, 물건, 사건, 개념)
- **약한 엔터티**: 다른 엔터티에 종속적으로 존재 (예: 방 - 건물에 종속)
- **강한 엔터티**: 독립적으로 존재 가능

### 4.1.2 릴레이션

- **릴레이션**: 데이터베이스에서 정보를 구분하여 저장하는 기본 단위
- 관계형 DB에서는 **'테이블'**, NoSQL에서는 **'컬렉션'**

### 4.1.3 속성(attribute)

- **속성**: 릴레이션에서 관리하는 구체적이며 고유한 이름을 갖는 정보

### 4.1.4 도메인(domain)

- **도메인**: 속성이 가질 수 있는 값의 집합 (예: 성별 → 남, 여)

### 4.1.5 필드와 레코드

- **필드**: 테이블의 열(column)
- **레코드/튜플**: 테이블의 행(row)

### 4.1.6 관계

- **1:1 관계**: 유저 ↔ 유저 이메일
- **1:N 관계**: 유저 → 상품(장바구니)
- **N:M 관계**: 학생 ↔ 강의 (중간 테이블 필요)

### 4.1.7 키

- **기본키(PK)**: 유일성과 최소성을 만족하는 키
- **외래키(FK)**: 다른 테이블의 기본키를 참조
- **후보키**: 기본키가 될 수 있는 후보
- **대체키:** 후보키가 두 개 이상일 경우 어느 하나를 기본키로 지정하고 남은 후보키들을 말함
- **슈퍼키**: 유일성을 갖춘 키

## 4.2 ERD와 정규화 과정

### 4.2.1 ERD의 중요성

- 데이터베이스 구축의 기초 설계도 역할
- 시스템 요구사항을 기반으로 작성

### 4.2.3 정규화 과정

- **목적**: 데이터베이스 이상 현상 해결, 저장 공간 효율성
- **제1정규형**: 원자값으로만 구성 (반복 집합 제거)
- **제2정규형**: 부분 함수 종속성 제거
- **제3정규형**: 이행적 함수 종속 제거
- **BCNF**: 모든 결정자가 후보키인 상태

## 4.3 트랜잭션과 무결성

### 4.3.1 트랜잭션

**ACID 특징**:

- **원자성(Atomicity)**: All or Nothing 트랜잭션과 관련된 일이 모두 수행되었거나 되지 않았거나를 보장
- **일관성(Consistency)**: 허용된 방식으로만 데이터 변경
- **격리성=독립성(Isolation)**: 트랜잭션 간 간섭 방지
  - **격리 수준**
    - SERIALIZABLE > REPEATABLE_READ > READ_COMMITTED > READ_UNCOMMITTED
    - 격리성 ↑, 동시성 ↓
- **지속성(Durability)**: 성공한 트랜잭션의 영구 반영(데이터베이스에 시스템 장애가 발생해도 원래 상태로 복구하는 회복 기능이 있어야함)
  - 체크섬 : 중복 검사의 한 형태로, 오류 정정을 통해 송신된 자료의 무결성을 보호하는 방법
  - 저널링: 파일 시스템 또는 데이터베이스 시스템에 변경 사항을 반영(commit)하기 전에 로깅하는 것
  - 롤백

### 4.3.2 무결성

- **개체 무결성**: 기본키는 NULL 불가
- **참조 무결성**: 참조 관계 데이터의 일관성
- **고유 무결성**: 특정 속성의 고유성
- **NULL 무결성**: 특정 속성의 NULL 제약

## 4.4 데이터베이스의 종류

### 4.4.1 관계형 데이터베이스(RDBMS)

- **MySQL**: 가장 많이 사용, MyISAM/InnoDB 엔진
- **PostgreSQL**: VACUUM 기능, JSON 지원

### 4.4.2 NoSQL 데이터베이스

- **MongoDB**: JSON 기반, BSON 저장, ObjectID 자동 생성
- **Redis**: 인메모리, 키-값 저장, 캐싱/세션 관리

## 4.5 인덱스

### 4.5.1 인덱스의 필요성

- 데이터를 빠르게 찾기 위한 장치
- 책의 찾아보기와 동일한 개념

### 4.5.2 B-트리

- **구조**: 루트 노드 → 브랜치 노드 → 리프 노드
- **대수확장성**: 트리 깊이가 리프 노드 수에 비해 매우 느리게 성장하는 것을 의미
  - 기본적으로 인덱스가 한 깊이씩 증가할 때마다 최대 인덱스 항목의 수는 4배씩 증가

### 4.5.3 인덱스 만드는 방법

- MySQL
  - 클러스터형 인덱스
    - 테이블당 하나를 설정

        - primary key 옵션으로 기본키로 만들면 생성 가능
        - 기본키로 만들지 않고 unique not null 옵션을 붙이면 만들 수 있음
    - 세컨더리 인덱스
- MogoDB
    - 도큐먼트를 만들면 자동으로 형성

### 4.5.4 인덱스 최적화 기법

1. **인덱스는 비용**: 무분별한 생성 지양
2. **항상 테스팅**: explain() 함수 활용
3. **복합 인덱스 순서**: 같음 → 정렬 → 다중값 → 카디널리티

## 4.6 조인의 종류

- **내부 조인(INNER JOIN)**: 교집합
- **왼쪽 조인(LEFT JOIN)**: 왼쪽 테이블 기준
- **오른쪽 조인(RIGHT JOIN)**: 오른쪽 테이블 기준
- **합집합 조인(FULL OUTER JOIN)**: 합집합

## 4.7 조인의 원리

### 4.7.1 중첩 루프 조인(NLJ)

- 중첩 for문과 같은 원리
- 대용량 테이블에는 부적합

### 4.7.2 정렬 병합 조인

- 각 테이블을 정렬 후 조인
- 범위 비교 연산자 사용 시 효과적

### 4.7.3 해시 조인

- 해시 테이블 기반 조인7
- **빌드 단계**: 작은 테이블로 해시 테이블 생성
- **프로브 단계**: 큰 테이블에서 매칭되는 레코드 검색
- 동등(=) 조인에서만 사용 가능

**▼ 예상질문**

▼ 인덱스를 사용하면 조회 성능이 향상되는데, 왜 모든 컬럼에 인덱스를 생성하지 않나요?

인덱스는 조회 성능을 향상시키지만, 다음과 같은 비용과 단점이 있기 때문입니다.

저장 공간 비용/ 쓰기 성능 저하/이중 탐색 비용/ 메모리 사용량 증가

**1. 저장 공간 비용**

- 인덱스는 별도의 저장 공간을 필요로 하며, 데이터가 많을수록 상당한 공간을 차지합니다.

**2. 쓰기 성능 저하**

- INSERT, UPDATE, DELETE 시 인덱스도 함께 수정해야 하므로 쓰기 성능이 저하됩니다.
- 특히 B-트리 구조의 균형을 맞추기 위한 추가 작업이 필요합니다.

**3. 이중 탐색 비용**

- 인덱스를 사용할 때 인덱스 테이블을 먼저 확인한 후 실제 데이터 테이블에 접근하는 두 번의 탐색이 필요합니다.

**4. 메모리 사용량 증가**

- 자주 사용되는 인덱스는 메모리에 캐시되어 메모리 사용량이 증가합니다.

▼ 트랜잭션의 ACID 특성에 대해 설명하고, 각각의 예시를 들어주세요.

트랜잭션의 ACID 특성은 데이터베이스의 무결성을 보장하는 네 가지 핵심 원칙입니다.

**원자성(Atomicity)** - 트랜잭션의 모든 작업이 완전히 수행되거나 전혀 수행되지 않음을 보장
**일관성(Consistency)** - 허용된 방식으로만 데이터 변경
**독립성(Isolation)** - 트랜잭션 간 간섭 방지
**지속성(Durability)** - 성공한 트랜잭션의 영구 반영

▼ 데이터베이스 정규화를 하는 이유를 설명해주세요.

정규화는 데이터베이스 이상 현상을 해결하고 저장 공간을 효율적으로 사용하기 위한 과정입니다.