프로그램, 프로세스, 스레드
프로그램은 파일 시스템에 존재하는 실행 파일
프로세스 실행되고 있는 프로그램의 인스턴스
IPC(Inter Process Communication) : 독립된 프로세스끼리 통신을 하기 위한 방법
1. 파이프(익명-서로 알기 때문) - 일방적 데이터 전송, 구현 간단하지만 일방적으로 송신
2. named pipe(FIFO) - 부모, 자식 프로세스 간의 통신 이외에도 사용 가능(이름 줘야 함), 파일을 이용하여 통신 -> 한 개는 읽기, 한 개는 쓰기로 이용 가능
3. Message Queue - 메모리 공간
4. 공유 메모리 - 메모리 영역을 공유한다. 가장 빠름(중개자가 없어서)
5. 소켓

각 스레드는 레지스터와 스택을 가진다. 이외의 모든 메모리를 공유한다.
context-switching : 멀티 프로세스에서 프로세스를 실행하고 있는 상태에서 다른 프로세스를 실행해야 할 때 기존 프로세스의 상태 및 레지스터 값을 저장하고 다음 프로세스를 위해 상태 및 레지스터 값을 교체하는 작업 -> 이 정보들은 PCB에 저장된다.
context-switching이 발생할 때 : 1. I/O 2. CPU사용 시간 만료 3. 자식 프로세스 생성 4. 인터럽트
os 스케쥴러 : 라운드 로빈(공평한 시간)

데드락의 발생 조건
1. 상호 배제 : 자원은 한 프로세스만 사용할 수 있다.
2. 점유 대기 : 하나의 자원을 점유하고 이미 사용되고 있는 자원을 점유하기 위해 대기하는 프로세스 있어야 함.
3. 비선점 : 다른 프로세스에 할당된 자원은 빼앗을 수 없다.
4. 순환 대기 : 각 프로세스는 서로 점유한 자원을 요구해야함.

해결 방법
1. 예방 : 발생 조건 중 하나를 제거 -> 자원 낭비가 심하다.
2. 회피 : 교착 상태를 피한다. 자원을 할당해도 안정 상태인지 사전에 검사한다.
3. 탐지 및 회복 : 교착 상태를 검사하고 1) 프로세스 모두 중지 2) 한 프로세스씩 중지

세마포어 및 뮤텍스
IPC를 이용할 경우 동기화 문제 발생
크리티컬 섹션 : 각 프로세스에서 데이터를 공유하는 구간
세마포어 : 변수만큼 프로세스or스레드가 접근 가능. 타인이 해제 가능(리더-라이터문제)
뮤텍스 : 오직 1개의 프로세스or스레드 접근 가능. 타인이 해제 불가

멀티 스레드 문제점
1. 사용 중인 변수나 자료 구조에 접근하여 동기화 오류 발생 가능
2. 동기화 처리 때문에 병목 현상이 발생하여 성능 저하

운영체제 : 시스템의 자원과 동작을 관리하는 소프트웨어

스택 메모리 : 컴파일 시에 크기가 결정되며 함수의 호출 시 할당되며 호출 완료시 소멸한다.
힙 메모리 : 런타임 시에 크기가 결정되며 사용자에 의해 동적으로 할당되고 해제된다.
속도는 스택 > 힙(할당을 해야함)

Blocking IO : 호출된 함수가 작업을 끝낼 때까지 제어권을 가지고 있어서 호출한 함수를 대기시킨다.
NonBlocking IO : 호출된 함수가 바로 제어권을 return하여 호출한 함수는 다른 일을 할 수 있다.

동기 / 비동기 : 호출되는 함수의 작업 여부를 누가 신경쓰느냐?
동기 : 호출된 함수의 수행 결과 및 종료를 호출한 함수가 신경쓴다.
비동기 : 호출된 함수의 수행 결과 및 종료를 호출된 함수 혼자 신경쓰고 처리(이벤트)
https://musma.github.io/2019/04/17/blocking-and-synchronous.html

메모리 관리 전략
페이징 : 논리 -> 페이지, 물리 -> 프레임으로 나눈다. 내부 단편화 발생
세그멘테이션 : 서로 다른 크기의 논리 단위인 세그먼트로 분할. 외부 단편화 발생

가상 메모리 : 프로세스의 필요한 부분을 메모리에 적재하는 것 -> valid bit 메모리에 있으면 1 없으면 0
만약 valid bit가 0일 때 인터럽트를 통해 디스크에 접근하여 해당 부분을 찾아서 메모리에 올린다.
페이지 교체 알고리즘 : LRU(Least Recently Used) - 가장 오랜 기간 사용되지 않은 페이지를 교체

캐시 메모리 : CPU와 메모리의 속도 차이로 인한 병목 현상을 완화하기 위해 사용
1. 시간 지역성 (temporal) - 한 번 참조한 데이터는 다시 참조할 가능성이 높다.(순환, 서브루틴, 스택)
2. 공간 지역성 (spatial) - 참조한 데이터와 인접한 데이터가 참조될 가능성이 높다.(배열, 순차 코드)
3. 순차 지역성 (sequential) - 분기가 발생하지 않는 한 명령어는 메모리의 순서대로 실행된다.

커널 모드 / 유저 모드 - 커널이 실행되어야 하는 경우 커널 모드로 전환, 기본적으로 유저 모드
유저 모드에서 동작할 때 커널 영역으로의 접근 금지. 모드의 전환은 시스템에 부담을 준다.

커널 레벨 쓰레드 : 스케줄링하는 주체가 커널. 장) 안정성과 다양한 기능. 단) 모드 전환으로 성능 저하.
유저 레벨 쓰레드 : 제공하는 기능 이외의 쓰레드. 장) 모드 전환이 없다. 단) 커널에 의해 블로킹 되면 프로세스 전체가 블로킹된다.

스와핑 : 주기억장치에 적재한 하나의 프로세스와 보조기억장치에 적재한 다른 프로세스의 메모리를 교체하는 기법. 단위는 프로세스이다.
가상 메모리는 페이지 단위로 교체한다.
메모리에 적재하면 Swap In, 보조기억장치로 내보내는 것을 Swap Out

Thrashing(쓰레싱) : 메모리 영역에 접근하게 될 때, 페이지 폴트가 높아 페이지 교체 시간이 많아짐
원인 : 멀티 프로그래밍 정도가 높아지고 CPU 이용률이 최대값에 도달하기 때문(각 프로세스의 요구되는 프레임의 개수보다 프레임이 적어짐) -> 프로세스가 수행되는 시간보다 페이지 교체 시간이 더 많아짐

Working Set : 프로세스에 의해 자주 참조되는 페이지들의 집합

장기 스케쥴러 : 대용량 메모리(디스크)에 있는 프로세스 중 어떤 프로세스에 메모리를 할당할지 정함
중기 스케쥴러 : 프로세스를 메모리에서 쫓아냄(ready->suspend)
단기 : cpu와 메모리 사이의 스케쥴링, 어떤 놈을 ready->running

OSI 7 계층 : 통신이 일어나는 과정을 단계별로 파악할 수 있다. 사람이 이해하기 쉽다.

1. 물리 계층 : 통신 케이블로 데이터를 전송, 통신 단위는 비트 (리피터, 허브, 케이블)

2. 데이터 링크 계층 : IP 주소를 이용 송, 수신되는 데이터의 오류와 흐름을 관리. 통신 단위는 프레임. 오류 검출(패리티, CRC), 재전송(Stop and Wait, Go-back-N), 흐름 제어(Stop and Wait, 슬라이딩 윈도우). 맥 주소를 이용한 통신 (브릿지, 스위치)

3. 네트워크 계층 : 데이터를 목적지까지 안전하고 빠르게 전달하는 기능(라우팅). 통신 단위는 패킷
   라우팅(거리 벡터-방향, 링크 상태-다익스트라), 혼잡 제어(다시 라우팅, 트래픽 성형-리키 버킷), 패킷 분할-병합(크기를 나눈다-단편화) (L3 스위치, 라우터)

4. 전송 계층 : 신뢰성 있는 통신을 목표로 함.
   통신 단위는 TCP-Segment, UDP-Datagram. (게이트웨이)

5. 세션 계층 : 데이터가 통신하기 위한 논리적인 연결. 세션을 설정 및 해제 통신 단위는 메세지

6. 표현 계층 : 데이터 표현이 상이한 응용 프로세스의 독립성을 제공, 암호화. 통신 단위는 메세지

7. 응용 계층 : 응용 프로세스와 직접 관계하여 일반적인 응용 서비스 수행. 통신 단위는 메세지

HTTP 프로토콜
1. 요청 및 응답 구조 : 서버/클라이언트 구조
2. 메시지 교환 프로토콜 : 클라이언트-서버 간에 HTTP메세지를 주고 받음. 응답 및 요청 메시지
3. 트랜잭션 중심의 비연결성 프로토콜 : (1) Connectionless (2) Stateless

TIME_WAIT
1. 지연 패킷으로 인한 오작동 방지
2. 클라이언트의 마지막 ACK신호 유실 시 새 연결 실패 방지

로드 밸런싱 : 트래픽이 몰리는 것을 방지하기 위해 각 서버로 분산시키는 것
1. 라운드 로빈 : 서버에 들어온 요청을 순서대로 돌아가며 배정
2. 가중 라운드 로빈 : 서버마다 가중치를 매기고 높은 서버에 우선 배분
3. IP 해시 : IP 주소를 특정 서버로 매핑하여 요청 처리
4. 최소 연결 : 요청이 들어온 시점에 가장 적은 연결 상태를 보이는 서버에 배분
5. 최소 리스폰타임 : 서버의 현재 연결 상태와 응답시간을 고려하여 트래픽 배분

L4 로드밸런싱 : 3, 4 계층의 정보를 바탕으로 분산(IP, Port, MAC, Protocol 등)
속도, 효율 좋음. 안전하다. 섬세하지 않다.
L7 로드밸런싱 : 사용자의 요청을 기준으로 분산(HTTP 헤더, 쿠키 등)
캐싱 기능, 비정상적 트래픽 차단. 보안 위험(클라 인증서 LB로 간다)

NAT(네트워크 주소 변환) : IP주소와 포트 번호를 재기록 - NAT Table
(IPv4 한계로 인해) -> L/B 가능

DNS 작동 순서(naver.com) - Recursive Query
Local DNS - Root DNS - com DNS - naver.com DNS

리피터(1) : 단순히 전기적인 신호 증폭, 연결된 모든 PC에 신호 전달
허브(1) : 리피터 + 패킷 모니터링, 멀티 포트 지원 -> 문제 생긴 곳 고립
브릿지(2) : 전송 거리 연장, 프레임을 다시 만들어 전송.
스위치(2) : 목적지 MAC주소를 가진 장비가 연결된 포트로만 프레임을 전송. 브로드캐스팅->성능저하
브릿지-스위치 : 스위치가 속도, 기능이 더 좋음
L3스위치 : 브로드 캐스팅 트래픽이 증가하기 때문에 VLAN 적용.
VLAN : 물리적인 스위치에서 논리적으로 가상의 LAN을 만들어서 통신함 (브로드캐스팅 성능 개선)
게이트웨이 : 외부로 연결되는 통로, 로컬망 라우터와 외부 망 라우터간의 통로.

HTTP 1.0
- 연결을 할 때마다 3way-handshaking
- 데이터 전송 완료시 바로 연결을 끊는다

HTTP 1.1
- 연결당 하나의 요청과 응답을 처리 -> 속도 및 성능 저하
- HOL Blocking : 클라이언트의 요청과 서버의 응답 순서는 동기화돼야 함.
- RTT 증가(양방향 지연)
- 헤더가 크다(쿠키) : 매 요청 시 중복된 헤더 값을 전송 (쿠키 포함)

HTTP 2.0
- 연결당 여러 개의 메시지를 동시에 주고 받을 수 있음
- 요청이 커넥션 상에서 다중화되므로 HOL Blocking은 발생하지 않음.
- Header 정보를 압축하여 전송

KeepAlive : 자동으로 연결이 해제되는 것을 막기 위함
TCP : payload가 없는 패킷을 주기적으로 보낸다.
HTTP : keepalive timeout 내에 재요청 시 열려 있는 커넥션을 통해 전송.

GET = url 뒤의 쿼리스트링을 통해 전송
POST = HTTP의 body에 담아 전송, 대용량 전송 가능

TCP 흐름 제어 : 수신 측의 버퍼가 가득차는 것을 방지(데이터 손실, 재전송으로 인한 낭비 방지)
1. Stop and Wait : 하나씩 전송하며 ACK를 받아야만 다음 전송
2. 슬라이딩 윈도우 : 윈도우 크기만큼 전송, ACK시 그만큼 이동

혼잡 제어 : 라우터에 데이터가 몰리는 것을 방지
1. AIMD : 1씩 증가시키고 혼잡 시 절반으로 줄인다.
2. Slow Start : +1, +2, +4, +8로 윈도우를 증가시키다가 혼잡시 1씩 증가
3. Fast Recovery : 혼잡 시 윈도우 반으로 줄이고 선형 증가

DHCP : 동적으로 IP 주소를 할당해주는 프로토콜(임대 개념)
장점 : PC를 켠 사용자만 IP할당 해 IP절약. DHCP서버만 변경하면 동적으로 변경 가능
단점 : 초기 부팅시 broadcast 트래픽 유발, Lease Time 까지 IP가 할당되지 못해 낭비
서버 의존이 크다

Primary Key, 기본키 : 후보키 중 선택한 키 (Null, 중복 안됨)
Candidate Key, 후보키 : tuple을 유일하게 식별하기 위해 사용하는 키(기본키 후보) - 유일성, 최소성
Alternate Key, 대체키 : 후보키 중 기본키를 제외한 나머지 키
Super Key, 슈퍼키 : 유일성은 만족하지만, 최소성은 만족하지 못하는 키

RDB
1. 스키마를 가진다.
2. 관계를 가진다.(데이터의 중복 제거)
3. 트랜잭션, 정규화

장점 : (1) 정렬, 탐색, 분류가 빠름 (2) 데이터의 무결성 보장 (3) 정규화로 갱신 비용 최소화
단점 : (1) 작성된 스키마 수정이 어렵다. (2) 빅데이터를 처리하는데 비효율적
용도 : 데이터가 자주 변경되는 경우, 명확환 스키마를 요구하며 구조가 변경되지 않을 때

NoSQL
1. 스키마에 대한 정의가 없다.
2. PK, FK, JOIN 등 관계가 없다.

장점 : (1) RDB에 비해 빠르다. (2) 데이터 모델링이 유연 (3) 복잡한 구조를 표현 가능
단점 : 쿼리 처리 시 파싱 후 연산을 해야해서 큰 크기의 document를 다루면 성능 저하
용도 : 데이터 요구사항을 알 수 없고, 관계를 맺는 데이터가 자주 변경되는 경우
읽기는 자주 하지만, 데이터가 자주 변경되지 않는 경우
DB를 수평으로 확장하는 경우 (많은 양의 데이터를 다루는 경우, 읽기/쓰기 처리량이 큰 경우)

NoSQL은 수평 확장이 유리하다.(샤딩)
게시물 + 댓글 저장 -> JOIN이 필요 없다.

INNER JOIN : A, B의 교집합
LEFT OUTER JOIN : A의 값
RIGHT OUTER JOIN : B의 값
FULL OUTER JOIN : A, B의 합집합
CROSS JOIN : 모든 경우의 수(A-3, B-4 => 12개)
SELF JOIN : 자기 자신과 조인 (A-3 => 9개)

트랜잭션 : 데이터베이스의 상태를 변화시키기 위해 수행하는 작업 단위
원자성(Atomicity) : 트랜잭션은 모두 반영되거나, 전혀 반영되지 않아야 한다.
일관성(Consistency) : 트랜잭션의 작업 처리는 항상 일관성 있어야 한다.
독립성(Isolation) : 트랜잭션은 다른 트랜잭션의 연산에 끼어들 수 없다.
지속성(Durability) : 트랜잭션이 성공적으로 완료되었으면, 결과는 영구적으로 반영돼야 한다.
Commit : 트랜잭션이 성공적으로 끝났을 때 알려주기 위해 사용하는 연산
Rollback : 트랜잭션이 비정상적으로 종료되어 원자성이 깨진 경우 되돌릴 수 있다.
버퍼 관리자에서 진행
UNDO : 트랜잭션 비정상적 종료 시 롤백, 복구
REDO : 적용된 변경사항에 대한 이력을 저장하여 다시 실행

데이터 사전(data dictionary) : 시스템 카탈로그, 데이터에 관한 정보를 저장한다.(메타 데이터)

Shared Lock : 읽기 작업을 할 때 거는 락 (여럿이서 가능)
Exclusive Lock : 쓰기 작업을 할 때 거는 락 (혼자만 가능)
LOCK의 문제점 : 1. 트랜잭션의 직렬화 2. 데드락 발생 위험(탐지, 회피)

정규화 : 데이터베이스의 이상 현상과 중복을 최소화하기 위해 분리하는 작업.
이상 현상(종속으로 인해 발생) : 삽입 이상, 갱신 이상, 삭제 이상

ORM
장점 : (1) 객체지향적인 코드로 더 직관적이다 (2) 재사용 및 유지보수 용이 (3) DBMS 종속성 줄어듬
단점 : (1) ORM의 한계가 있다. (2) 프로젝트가 복잡하면 구현하기 힘들다.

무결성 : 데이터의 정확성, 일관성, 유효성을 유지하는 것.
1. 개체 무결성 : 기본키는 NULL, 중복 허용 X
2. 참조 무결성 : 외래키는 NULL, 부모 테이블 기본키에 종속
3. 도메인 무결성 : 올바른 데이터 타입이 입력되었는지 확인
4. 고유 무결성 : 해당 속성 값은 모두 고유한 값을 가짐
5. NULL 무결성 : 특정 속성 값에 NULL 될 수 없는 제약 조건
6. 키 무결성 : 각 릴레이션에는 최소한 한 개의 키가 존재해야 한다

Index : RDBMS에서 검색 속도를 높이기 위한 기술
Column을 색인화 -> B+ Tree 구조로 Full Scan 하지 않음
단점 : 데이터 변경 작업이 자주 일어나는 경우, Index를 재작성해야 하므로 성능에 영향을 미침.
Delete 시 데이터를 지우지 않고 사용 안함으로 표시하기 때문에 Table의 데이터와 Index의 데이터가 다를 수 있다.

인덱스 종류
1. B+ Tree
2. Bitmap - 비트를 이용해 컬럼값을 저장하고 인덱스를 생성(B+ Tree는 값을 다 가지고 있어야 함)
3. 함수 인덱스
4. 해시 인덱스 - 빠르지만 등가비교만 가능(기본키 유리)
5. 풀 텍스트 인덱스 - 구분자, N-그램(글자수로 끊음) 기법
6. R 트리 인덱스 - 2차원 공간 개념 인덱스

해시 충돌 해결 방법(비둘기집 원리)

1. Separate Chaining (분리 연결법) : 링크드 리스트를 이용 (메모리 문제)
2. 보조 함수
3. Open Addressing (개방 주소법) - 연결 리스트, 트리 : 다른 인덱스에 저장
4. 버킷의 사이즈 증가(75%)

객체 지향 : 특정한 개념의 함수와 자료형을 함께 묶어서 관리하는 것

1. 추상화 : 필요로 하는 속성이나 행동을 추출하는 것 (BMW, 벤츠 -> 자동차)
2. 캡슐화 : 낮은 결합도를 유지할 수 있도록 설계하는 것 -> 정보 은닉
3. 상속 : 개체들이 지닌 공통된 특성을 하나의 개념이나 법칙으로 성립하는 것 (캡슐화와 비슷)
   단점 (1) 상위 클래스 변경이 어려워짐 (2) 불필요한 클래스 증가
4. 다형성 : 다른 클래스의 객체가 같은 메시지를 받았을 때 각자의 방식으로 동작하는 것

설계 원칙(SOLID Principle)

1. SRP(Single Responsibility) - 단일 책임 원칙 : 클래스는 단 한 개의 책임을 가져야 한다.
2. OCP(Open-Closed Principle) - 개방, 폐쇄 원칙 : 확장에는 열려 있어야 하고, 변경에는 닫혀 있어야 한다.
3. LSP(Liskov Substituion) - 리스코프 치환 원칙 : 상위 타입의 객체를 하위 타입의 객체로 치환해도 정상 동작해야 한다.
4. ISP(Interface Segregation) - 인터페이스 분리 원칙 : 인터페이스는 사용하는 클라이언트 기준으로 분리해야 한다.
5. DIP(Dependency Inversion) - 의존 역전 원칙 : 고수준 모듈은 저수준 모듈에 의존해서는 안된다.

직렬화 : 객체 또는 데이터를 외부의 자바 시스템에서도 사용할 수 있도록 바이트 형태로 데이터 변환하는 기술과 이 데이터를 다시 객체로 변환하는 기술 (csv, json, binary)

.java ->(컴파일러) .class, 바이트코드 ->(JVM) 기계어 -> 인터프리터 방식 실행

L4 : 응용 계층 - FTP, Telnet, SSH
L3 : 전송 계층 - 통신 노드 간 연결 제어, 신뢰성 있는 데이터 전송
L2 : 인터넷 계층 - IP 패킷 전송, 라우팅
L1 : 네트워크 엑세스 계층 - MAC 이용

스풀링 : 디스크를 매우 큰 버퍼로 이용, 다수 프로세스가 I/O를 요구하거나 장치의 수가 제한된 경우 이를 공유하기 위한 가상 장치를 프로세스에게 제공해주는 개념

라우터 : 수신한 패킷의 정보를 통해 경로를 설정해 패킷을 전송
스위치 : 내부 네트워크에서 MAC 주소를 이용해 해당 프레임을 전송

REST API : HTTP를 통해 CRUD를 실행하는 API (JSON, XML), 자원은 각각의 URI를 가진다.
장점 : HTTP 프로토콜을 이용하므로 따로 인프라가 필요 없다. HTTP를 사용하는 모든 플랫폼 호환.
단점 : 표준이 없다. 메소드가 4개(GET, POST, DELETE, PUT) 밖에 없다.

GraphQL : Query Language를 이용해 Server API의 정보를 주고 받기 위한 언어
1개의 엔드포인트를 가진다. Query문에 따라 응답의 구조가 달라진다. 유연하다.
장점 : HTTP요청의 횟수를 줄일 수 있다. 원하는 정보를 하나의 Query에 담을 수 있다.
단점 : Text만으로 하기 힘든 내용을 처리하기 힘들다. 재귀적인 Query가 불가능하다.

ERP : 전사적 자원관리 (기업 내의 모든 자원을 관리하는 통합정보시스템)

서버 장애시 대응
1. 롤백 - 데이터 스키마 변경 등 롤백 시 문제를 초래할 수 있다.
2. 서버 재기동 - 트래픽 급증, 코드 문제로 인한 리소스 부족
3. 장비 증설 - 트래픽 급증, 미리 장비 증설하기 쉬운 구조로 만들어야 한다.
4. 핫픽스 배포 - 문제를 수정한 버전 배포. 롤백, 재기동이 어려울 때.

절차 지향 언어(C) : 순차적인 처리로 컴퓨터의 작업 방식과 유사하여 빠르다
장 : 빠르다 단 : 유지보수, 디버깅이 어렵다. 실행 순서가 정해져 있다.

객체 지향 언어(Java) : 현실을 모델링하여 개발하는 것
장 : 코드의 재활용이 쉽다. 디버깅이 쉽다. 단 : 처리속도가 느리다. 설계에 시간이 소요된다.

인터프리터 언어(Python) : 컴파일과 실행이 동시에 이루어지는 언어

캐시 메모리 구조
1. direct mapped
2. fully associative
3. set associative

클러스터 인덱스
1. 인덱스 생성시 데이터 페이지 재정렬
2. 검색 속도가 논클러스터보다 빠르지만, 입력/수정/삭제가 느리다.

논클러스터 인덱스
1. 별도의 페이지에 인덱스 구성
2. 검색 속도는 느리지만, 입력/수정/삭제가 빠르다.