CS 기술 면접 대비

Section 01

Design Patterns & Programming Paradigm

Library와 Framework를 설명

* Library와 Framework 모두 특정한 기능들을 모듈화한 미리 작성된 코드이다. flow의 제어권이 프로그래머에게 있는지 아니면 framework에 있는지로 Library와 Framework가 구별된다.

Singleton Pattern

* 클래스 하나에 오직 하나의 인스턴스를 가지도록 하는 패턴.
* 인스턴스를 마구 생성하는 것에 비해 여러 비용(메모리와 속도)측면에서 이점을 가질 수 있음
* 하나의 인스턴스이기에 객체 상태의 공유가 유의미한 환경에서 잘 사용된다.
* 다만 TDD의 걸림돌이 될 수 있다. 테스트는 서로 독립적이어야 하고 어떤 순서로든 실행될 수 있어야 하는 반면 싱글톤 패턴은 미리 생성된 하나의 인스턴스를 기반으로 동작하기에 독립적인 인스턴스를 생성하기 어려움.
* 하지만 싱글톤 내에 상태를 초기화하는 메서드를 하나 구현하면 독립적으로 사용할 수 있을 것.
* 또 다른 싱글톤 패턴의 단점은 모듈간의 결합을 강하게 만든다는 것. 이것은 의존성 주입을 통해 모듈간의 결합을 조금 더 느슨하게 만들어 해결할 수 있음

Simple Factory Pattern

* 객체의 생성 부분을 떼어내어 추상화한 패턴. (객체의 생성을 담당하는 Factory 클래스가 있다)
* 다음은 Simple Factory Pattern의 모습

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* SimplePhoneFactory에서 type을 따져 휴대폰의 객체를 직접 생성하는 것을 볼 수 있음
* 객체 생성의 세부 사항을 Phone을 구현하는 각 클래스에 두고 객체를 만드는 작업을 하나의 팩토리 클래스에 모아둔 모습을 볼 수 있음 (생성하는 부분만을 떼어내어 Factory 클래스에 두었다)

Factory Method Pattern

* 객체를 생성하기 위한 팩토리 인터페이스를 정의, 어떤 클래스의 인스턴스를 만들지는 팩토리를 구현하는 서브클래스에서 결정하도록 하는 패턴
* 다음은 Factory Method Pattern의 모습

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* PhoneFactory를 상속, 혹은 구현한 서브 클래스에서 객체를 만들게끔 하는 것이 팩토리 메서드 패턴
* 구체적인 클래스 대신 인터페이스 메서드를 통해 phone을 생성 가능 유연하고 확장성 높다

Abstract Factory Pattern

* 구체적인 클래스에 의존하지 않고 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공
* 예로 IphoneFactory에서는 폰을 만들 때 createOS를 통해 IOS를 넣고 AndroidPhoneFactory에서는 createOS를 통해 AndroidOS를 넣으면 객체의 조합을 반영하여 생성 로직을 분리한 케이스가 된다.
* 즉 PhoneFactory는 조합하는 구현체를 제공하는 것

Strategy Pattern

* 캡슐화된 알고리즘을 갈아낄 수 있도록 함으로써 코드의 동작을 다양화하는 패턴. 다형성을 이용해서 구현할 수 있다. OCP를 지킬 수 있음

의존성 주입

* 의존성 : A가 B에 의존성이 있다는 말은 B의 변경에 A가 반드시 변해야 함을 의미
* 의존성 만을 주입하는 중간자(Ioc Container)에 필요한 모듈들을 등록하고 의존성을 주입시키는 방법. (개발자가 직접 주입하는 것이 아니라 중간자가 개입함으로 제어의 역전이 일어남) 모듈 간의 결합을 느슨히 하여 모듈간 전환을 부드럽게 하며 SRP OCP를 위배하지 않도록 도움을 준다.

Observer Pattern

* 관찰자가 객체의 상태변화를 관찰하고 상태가 변화할 때 마다 옵저버들에게 변화 상황을 알리는 패턴

자바 상속 vs 구현

* 상속: 자식 클래스가 부모 클래스의 속성과 메서드를 상속받아 추가 및 확장을 가능하도록 하는 기능. 재사용성, 중복성의 최소화를 통해 객체지향의 특징을 강화해준다.
* 구현: 부모 인터페이스를 자식 클래스에서 재정의하여 구현하는 것을 말함 상속과 달리 반드시 부모 클래스의 메서드를 재정의해야 함
* 자바에서는 diamond problem으로 다중 상속을 허용하지 않는다. 하지만 인터페이스는 다중 구현이 가능함 이것은 자바에서의 상속과 구현의 차이
* Diamond problem
  + 텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

    자동 생성된 설명
  + Child는 Father를 extends하고 Mather도 extends 다만 Person의 특정 메소드가 Father와 Mother에서 다르게 구현되어 있다면 child는 Person의 특정메소드를 실행할 때 Father 것으로? Mother의 것으로? -> diamond problem

프록시 패턴, 프록시 서버

* 프록시 패턴이란 대상 객체의 앞단에 흐름을 가로챌 수 있는 가상 객체를 두어 인터페이스 역할을 할 수 있도록 설계한 패턴을 말함.
* 이를 통해 프록시 서버를 만들 경우 보안적인 측면에서 공격의도가 있는 요청을 걸러낼 수도 있고 CDN등을 구축하여 서버 응답성을 높일 수도 있음
* 또한 모든 요청을 곧이곧대로 서버에 보내면 SPOF 문제를 직면할 수 있는데에 반해 프록시 서버를 만들 경우 SPOF 문제를 해결할 수 있음.

DDOS 공격

* Distributed Denial Of Service: 공격대상이 되는 서버의 서비스 중단을 목적으로 표적 서버에 대량의 트래픽을 유발하는 공격
* 프록시 패턴을 이용한 CloudFlare를 이용하면 사용자가 아닌 시스템으로부터 오는 트래픽을 자동으로 차단하여 공격을 쉽게 막아낼 수 있다 (프록시 패턴 이용 예시)

CDN (Contend Delivery Network)

* 분산된 형태의 서버 네트워크를 말함 콘텐츠를 사용자에게 가까운 여러 프록시 서버에 데이터를 캐싱, 배포해놓음으로써 요청에 대한 응답성을 높일 수 있음(SPOF 문제 해결)
* 이러한 특징 때문에 주로 CDN은 한번 배포하면 수정이 적은 특징의 데이터를 위한 서버 네트워크를 구축할 때 사용이 됨. 영상 컨텐츠 등이 그러하기에 넷플릭스 왓챠 등 여러 OTT에서 CDN이 사용되는 것을 확인할 수 있음.
* CDN의 정책에는 여러가지가 있는데 사용자가 여러 서버 노드들 중 핑을 날려보고 응답성이 가장 좋은 프록시 서버에게 요청하도록 하는 정책을 NETFLIX에서 사용한다고 분산시스템 강의에서 배운 기억이 있음

Iterator Pattern

* Iterator 라는 인터페이스로 컬렉션을 순회할 수 있도록 설계한 패턴. 자료형에 구애 받지 않고 하나의 인터페이스로 순회가능한 것이 특징

Reveal Module Pattern

* 자바스크립트와 같은 언어는 접근 제어자가 없음 이러한 언어에서 접근 제어자 역할을 할 수 있는 것을 만드는 패턴

MVC Pattern

* 애플리케이션 구성요소를 Model, View, Controller로 나눈 패턴. 이렇게 나눔으로써 애플리케이션 개발 프로세스에서 코드 재사용성과 확장성을 높일 수 있다.
* Model : 어플리케이션의 데이터에 해당하는 부분(데이터 베이스, 상수, 변수 등등)
* View : 유저가 마주하게 되는 인터페이스 요소들
* Controller : 한 개 이상의 View와 한 개 이상의 Model을 연결하는 중간 다리 역할을 하는 요소

MVP Pattern

* MVC Pattern의 Controller 요소가 Presenter로 바뀐 패턴. Presenter는 Controller와 다르게 뷰와 Presenter가 1:1로 강한 결합을 가진다는 특징이 있음

MVVM Pattern

* MVC Pattern의 Controller 요소가 View Model로 바뀐 형태의 Pattern View Model은 View의 추상화된 버전으로 커맨드와 데이터 바인딩을 가지는 것이 MVC Pattern과 구별되는 차이점. 뷰와 뷰모델 사이의 양방향 데이터 바인딩을 지원한다.

Programming Paradigm

* Declarative 선언적 언어 (함수형 언어)
  + 컴퓨터에게 목적을 알린다. ”프로그램은 함수로 이루어진 것이다”라는 명제를 깔고 감
  + 함수형 언어에선 함수가 일급 객체다.
    - 일급 객체
      * 함수에 인자로 넘기기 가능
      * 변수나 메서드에 할당 가능
      * 함수가 반환 가능
      * 위 3가지 조건을 만족하는 객체는 일급 객체
  + 즉 함수형 언어에서 함수는 변수나 메서드에 할당 가능하고 함수가 함수를 반환가능하며 함수에 함수를 인자로 넘길 수 있다.
* Imperative 명령형 언어
  + 컴퓨터에게 알고리즘, 즉 어떻게 해야 할 지를 알린다. 변수의 값을 계속 수정하는 데에서 함수의 결과가 계속 바뀔 수 있는 Side Effect가 발생하는 것이 특징. 명령형 언어는 이러한 side effects를 십분 활용하는 패러다임이라고 볼 수도 있음
  + 하지만 요새는 람다와 같은 것들이 도입되면서 함수형 언어와 명령형 언어를 나누는 clear cut이 없음
* 절차형 언어
  + 수행되어야 할 연속적인 계산 과정으로 이루어짐
  + 일이 진행되는 방식으로 그저 코드를 구현하기만 하면 되기 때문에 코드의 가독성이 좋으며 실행 속도가 빠름
* 객체지향형 언어
  + 객체의 집합으로 프로그램의 상호작용을 표현하는 프로그래밍 패러다임
  + 데이터를 객체로 취급하여 객체 내부에 선언된 메서드를 활용하는 방식
  + 설계에 많은 시간이 소요되며 처리속도가 다른 프로그래밍 패러다임에 비해 느리지만 코드의 재사용성을 올려주어 개발 생산성을 많이 높임 (SW crisis를 객체지향으로 극복했다)
  + 객체지향 프로그래밍의 특징
    - 추상화 : 핵심적인 개념 또는 기능들을 간추리는 것을 의미
    - 캡슐화 : 객체의 속성과 메서드를 하나로 묶고 일부를 외부에 감추어 은닉하는 것
    - 상속성 : 상위 클래스의 특성을 하위 클래스가 이어 받아서 재사용하거나 추가 확장하는 것을 의미
    - 다형성 : 하나의 메서드나 클래스가 다양한 모습으로 동작하는 것을 의미 대표적으로 오버로딩과 오버라이딩이 다형성의 예이다.
      * 오버로딩: 같은 이름을 가진 메서드가 여럿 있을 수 있는 것을 의미. 메서드의 타입, 매개 변수 유형, 개수 등의 차이를 두며 여러 개를 둘 수 있으며 컴파일 중에 발생하는 정적 다형성
      * 오버라이딩: 상위 클래스의 메서드를 하위 클래스가 재정의하는 것을 의미 runtime 중에 발생하는 동적 다형성
  + 객체지향 프로그래밍의 설계 원칙
    - SRP (Single Responsibility Principle)
      * 모든 클래스는 각각 하나의 책임만을 가져야 하는 원칙
    - OCP (Open Closed Principle)
      * 확장에는 열려있고 변화에는 닫혀있어야 하는 원칙 즉, 기존의 코드는 잘 수정하지 않으면서 확장은 쉬워야 한다는 것
    - LSP (Liskov Substitution Principle)
      * 프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 하는 것을 의미. 부모 객체에 자식 객체를 넣어도 시스템이 문제없이 돌아가게 만드는 것을 뜻한다.
    - ISP (Interface Segregation Principle)
      * 하나의 일반적인 인터페이스보다 구체적인 여러 개의 인터페이스를 만들어야 하는 원칙
    - DIP (Dependency Inversion Principle)
      * 자신보다 변하기 쉬운 것에 의존하던 것을 추상화된 인터페이스나 상위 클래스를 두어 변하기 쉬운 것의 변화에 영향 받지 않게 하는 원칙, 즉 상위 계층은 하위 계층의 변화에 대한 수현으로부터 독립해야 한다. 추상화에 의존해야지 구체화에 의존하면 안된다!

Section 02

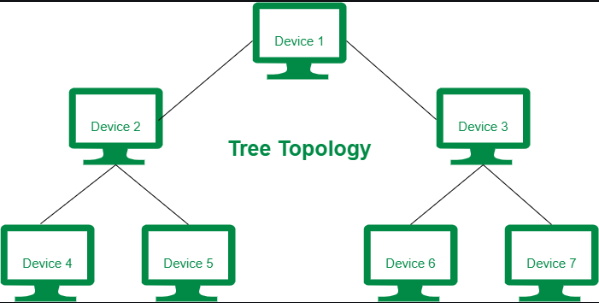
Network

네트워크

* 노드와 링크가 서로 연결되어 있으며 리소스를 공유하는 집합
* 서버, 라우터, 스위치 등이 노드에 속하며 유선 또는 무선 연결이 링크를 의미
* 네트워크를 구축할 때는 많은 처리량을 처리할 수 있으며 지연 시간이 짧고 장애 빈도가 적으며 좋은 보안을 갖춘 네트워크가 되도록 설계해야 함
* 처리량(throughput) : 시간당 처리되는 데이터양 단위로는 bps(bit per second)
* 지연 시간: 요청이 처리되는 시간 (요청자 입장에서 응답이 오는 데에 걸린 시간 -> 왕복 시간)
* 대역폭 : 주어진 시간 동안 네트워크 연결을 통해 흐를 수 있는 최대 비트 수
  + Cf> 대역폭이 넓으면 넓을수록 throughput과 latency가 좋아질까?
  + NO. 아니다. 대역폭이 넓은 상태에선 Symbol 길이를 짧게 설정할 수 밖에 없음 짧아진 Symbol은 frequency selectivity를 놓게 하고 ISI가능성을 높인다. 요새는 narrow band를 여러 개 사용하는 multicarrier modulation이 많이 쓰인다.

Network Topology (네트워크 연결 형식)

* Tree Topology



* + 계층형 토폴로지로 트리 형태의 네트워크 구성을 가짐
  + 장점 : 노드의 추가, 삭제가 쉬움
  + 단점 : 특정 노드에 트래픽이 몰릴 경우 하위 노드가 영향을 받을 수 있다.
* Bus Topology

컴퓨터, 스크린샷, 직사각형, 디자인이(가) 표시된 사진

자동 생성된 설명

* + 중앙 통신 회선 하나에 여러 개의 노드가 연결되어 공유하는 형태의 네트워크 구성. 주로 근거리 통신망(LAN)에서 사용
  + 장점 : 설치 비용이 적고 신뢰성 우수, 노드의 추가 삭제 쉬움
  + 단점 : 스푸핑에 취약 (Spoofing: 원래 목적지가 아닌 노드에서 패킷을 가로채는 것)
* Star Topology

컴퓨터, 디자인이(가) 표시된 사진

자동 생성된 설명

* + 중앙에 있는 노드에 다른 노드들이 모두 연결된 네트워크 구성
  + 장점 : 노드를 추가하거나 에러 탐지가 쉽고 패킷 충돌 발생 가능성 낮음
  + 단점 : 중앙 노드에 장애 발생하면 전체 네트워크 마비(SPOF)
* Ring Topology

텍스트, 도표, 스크린샷, 스케치이(가) 표시된 사진

자동 생성된 설명

* + 각 노드가 양옆으로 연결, 전체적으로 고리의 형태. 하나의 연속된 길을 통해 통신
  + 장점 : 노드 수가 증가되어도 네트워크 상의 손실이 거의 없고 충돌 빈도 낮음
  + 단점 : 네트워크 구성 변경 어렵고 회선 장애에 SPOF (노드는 분산되었지만 회선은 Single Point)
* Mesh Topology

도표, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명

* + 노드들이 그물망 처럼 연결되어 있는 형태
  + 장점 : 장애 노드 발생해도 여러 개의 경로가 존재함으로 장애에 강하며 트래픽 분산 처리 가능
  + 단점 ; 노드의 추가가 어렵고 구축과 운용 비용이 고가

규모 기반 네트워크 분류

* WAN (Wide Area Network) : 세계 규모
* MAN (Metropolitan Area Network) : 대도시 규모
* LAN (Local Area Network) : 건물 등 로컬 규모

네트워크 병목 현상을 만났을 때 문제를 확실히 짚기 위한 명령어들

* Ping: 네트워크 상태를 확인하려는 대상 노드를 향해 일정 크기의 패킷을 전송하는 명령

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

* + TCP/IP 프로토콜 중 ICMP 프로토콜을 기반으로 동작하기에 ICMP를 지원하지 않는 기기를 대상으로는 실행할 수 없음 마찬가지로 네트워크 정책상 traceroute를 차단하는 대상도 실행 불가
  + Cf> TTL(time to live) : IP 패킷 내에 있는 값으로써 패킷이 네트워크 내에 너무 오래 있어서 버려져야 하는지의 여부를 라우터에게 알린다.
    - 즉 정한 시간이 지나면 그 패킷을 버리고 재전송할 것인지를 결정하도록 발신인에게 알릴 수 있게 하기 위한 방법
    - 라우터에서 라우터로 네트워크 내의 무한정 잔류를 막기 위한 것
    - 라우터는 패킷을 수신할 때마다 TTL 수에서 하나를 빼고 네트워크의 다음 위치로 전달 TTL이 0이면 라우터는 패킷을 폐기, ICMP 메시지를 발신지 호스트로 보낸다.
* Netstat: 이용 중인 서비스들의 네트워크 상태를 표시하는 데 사용되며 네트워크 접속, 라우팅 테이블, 네트워크 프로토콜 등의 리스트를 보여준다.

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

* Nslookup : 특정 도메인에 매핑된 IP를 확인

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* Tracert(linux -> traceroute) : 목적지 노드까지 네트워크 경로를 확인할 때 사용하는 명령

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

네트워크 프로토콜 표준화

* 네트워크 프로토콜이란 다른 장치들끼리 데이터를 주고받기 위해 설정된 공통된 인터페이스(협약, 규약)
* IEEE, IETF등의 표준화 단체가 정한다.

TCP/IP 4계층 모델 / OSI 7계층

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* Application Layer
  + FTP, HTTP, SSH, SMTP, DNS 등 응용프로그램이 사용되는 프로토콜 계층이며 유저와 실질적으로 맞닿는 계층
  + FTP : File Transfer Protocol
  + SSH : Secure Shell 보안 프로토콜
  + HTTP : Hyper Text Transfer Protocol WWW를 위한 데이터 통신의 기초이자 웹 사이트를 이용하는 데 쓰는 프로토콜
  + SMTP : Simple Mail Transfer Protocol 전자 메일 전송 인터넷 표준 통신 프로토콜
  + DNS : 도메인 이름과 IP 주소를 매핑해주는 분산 서비스
* Tansport Layer
  + 송신자와 수신자를 연결하는 서비스 제공
  + TCP : 패킷 순서 보장, 연결지향, 가상회선 패킷 교환 방식
    - TCP가 Reliable하도록 하는 것은 3-way Handshake

텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

* + - 1단계 SYN : 클라이언트가 서버에 클라이언트의 ISN을 담아 SYN 전송 (ISN : 새로운 TCP 연결에 할당된 임의의 시퀀스 번호)
    - 2단계 SYN+ACK : 서버는 클라이언트의 SYN 수신, 서버의 ISN과 함께 클라이언트의 ISN + 1(승인 번호)을 보낸다.
    - 3단계 ACK : 클라이언트는 서버의 ISN + 1한 값인 승인번호를 담아 ACK를 서버에 전송
    - Cf> TCP 연결 해제 과정 4-way Handshake

텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진

자동 생성된 설명

* + - 1단계 FIN : 클라이언트가 연결을 닫으려고 할 때 FIN으로 설정된 세그먼트 전송
    - 2단계 ACK : 서버는 클라이언트로 ACK라는 승인 세그먼트 보내고 CLOSE WAIT 상태에 들어간다.
    - 3단계 FIN : 서버는 ACK를 보내고 일정 시간 후에 클라이언트에 FIN이라는 세그먼트 보냄
    - 4단계 ACK : 클라이언트는 서버로 ACK를 보내고 서버는 CLOSED 상태로 변경 이후 클라이언트는 어느 정도의 시간을 대기한 후 연결을 닫는다. (지연 패킷에 대비하기 위해 일정 시간 기다린다.)
  + UDP : 순서 보장 안함, 수신 여부 확인 X, 던지고 끝
* IP Layer
  + 장치로부터 받은 네트워크 패킷을 IP 주소로 지정된 목적지로 전송하기 위한 계층
* Link Layer
  + 전선, 광섬유, 무선 등으로 실질적으로 데이터를 전달하며 장치 간에 신호를 주고 받는 규칙을 정하는 계층
  + Full duplex(전 이중화 통신): 양쪽 장치가 동시에 송수신할 수 있는 방식
  + Cf> CSMA/CD: Half duplex의 일종 데이터를 보낸 후 충돌이 발생한다면 일정 시간 이후 재전송 하는 방식. (아마 3g였나 이전 세대에서 사용한 것으로 알고 있음)
  + Half duplex (반 이중화 통신): 양쪽 장치는 서로 통신할 수 있지만 동시에는 통신 불가능, 한 번에 한 방향만 통신할 수 있는 방식

PDU (Protocol Data Unit)

* 네트워크의 어떠한 계층에서 계층으로 데이터가 전달될 때 한 덩어리의 단위를 PDU라고 한다.
  + Application Layer에서 PDU는 Message
  + Transport Layer에서 PDU는 Segment(TCP) 혹은 Datagram(UDP)
  + Ip Layer에서 PDU는 Packet
  + Link Layer에서 PDU는 Frame(Link Layer) 혹은 bit(Physical layer)
* 계층 간 이동하면서 PDU의 헤더와 페이로드(데이터)가 계속 덧붙여 진다. (캡슐화) 상대편에서는 계층마다 비 캡슐화를 통해 데이터 처리

Network 기기들

* Application Layer를 처리하는 기기 : L7 switch (AKA Load Balancer)
  + 여러 장비를 연결하고 데이터 통신을 중재, 연결된 포트로만 전기 신호를 보내 데이터를 전송하는 장비
  + 로드밸런서라는 이명답게 서버의 부하를 분산하는 기기, 클라이언트의 요청을 뒤쪽의 여러 서버로 나누는 역할을 한다.
  + 높은 계층을 처리하는 기기이기에 URL, 서버, 캐시, 쿠키들을 기반으로 트래픽을 분산한다. 바이러스, 불필요한 외부 데이터등을 걸러내는 필터링 기능 또한 가지고 있으며 응용 프로그램 수준의 트래픽 모니터링도 가능하다.
  + L4 스위치 역시 로드밸런서로서의 역할을 수행하지만 L4스위치는 전송계층에서 동작하는 기기이기에 IP와 포트 기반으로 (특히 포트위주로) 트래픽을 분산한다.
  + Health Check: 로드밸런서는 주기적 헬스 체크를 통해 정상 서버와 비정상 서버를 판별하여 트래픽 분산에 참고한다. (요청 횟수가 적절해야 서버 부하와 장애 방치의 중간인 적절한 헬스 체크가 가능하다.)
  + 로드밸런서의 대표적 기능으로 서버 이중화를 들 수 있다. 서비스를 안정적으로 운용하기 위해 서버 다중화는 필수적인데 서버 다중화에서 로드 밸런싱이 빛을 발하는 것이다.
* Internet Layer를 처리하는 기기 : Router, L3 Switch
  + Router: 최적화된 경로로 패킷을 forwarding하는 routing을 수행하는 기기
  + L3 switch: L2 스위치의 기능과 라우팅 기능을 갖춘 장비 (사실상 라우터라고 보아도 무방)
    - L2 스위치는 MAC 주소 테이블을 참조하여 routing하는 스위치이고 L3 스위치는 routing table을 참조하여 IP주소를 기반으로 패킷을 forwarding하는 라우터이다. (간단 정리: IP 주소는 여러대의 컴퓨터가 공유하는 경우가 흔하지만 MAC주소는 장치 고유 주소)
* Data Link Layer를 처리하는 기기 : L2 Switch, Bridge
  + L2 Switch : MAC 주소(하드웨어 기반 물리적 주소, 변경 불가, 네트워크에 있는 고유한 장치를 식별하는데 사용된다.) 테이블을 통해 장치들의 MAC 주소를 관리하며 연결된 장치로부터 패킷이 왔을 때 패킷 전송을 담당, IP 주소를 이해하지 못해 IP 주소를 기반으로 라우팅은 불가능하며 단순히 패킷의 MAC 주소를 읽어 스위칭하는 역할을 수행한다.
  + Bridge: 두개의 근거리 통신망(LAN)을 상호 접속할 수 있도록 하는 통신망 연결 장치로, 포트와 포트 사이의 다리 역할을 하며 장치에서 받아온 MAC 주소를 테이블로 관리
* Physical Layer를 처리하는 기기: NIC, Repeater, AP
  + NIC: LAN 카드라고 하는 Network Interface Card(NIC)는 2대 이상의 컴퓨터 네트워크를 구성하는 데 사용하며 컴퓨터 내에 설치하는 확장 카드 (랜선 꽃는 구멍있고, 주민등록번호처럼 각각을 구분하기 위한 MAC 주소 있음)
  + Repeater: 약해진 신호를 증폭하여 다른쪽으로 전달하는 장치 but 요새는 광케이블 쓰면서 Deprecated
  + AP(Access Point) : 패킷을 복사하는 기기. 예로 AP에 유선 랜을 연결하고 무선 LAN 기술을 사용하여 무선 네트워크 연결을 할 수 있다.

IP 주소

* ARP (Address Resolution Protocol): IP 주소로부터 MAC 주소를 구하는 프로토콜 (MAC -> IP는 RARP)
  + ARP Request가 Broadcast를 보내어 IP에 해당하는 MAC주소를 찾는다. 해당 주소에 맞는 장치가 Unicast를 통해 자신임을 알린다.
* Hop by Hop communication
  + Ip주소를 통해 통신하는 과정은 홉바이홉
  + 패킷이 여러 개의 라우터를 건너가는 모습
  + 라우터들은 목적지 정보와 목적지로 가기 위한 방법이 들어있는 리스트인 Routing Table을 참조하여 홉바이홉 통신을 수행한다.
  + 게이트웨이는(라우터와 다름) 서로 다른 통신망, 프로토콜을 사용하는 네트워크 간의 통신을 가능하게 하는 관문 서로 다른 통신 프로토콜을 변환해주는 역할을 하기도 한다.
* IP 주소 체계
  + IPv4 : 32비트를 8비트 단위로 점을 찍어 표기
  + IPv6 : 64비트를 16비트 단위로 점을 찍어 표기
  + 추세는 IPv6로 가고 있지만 현재까지 많이 사용되는 주소 체계는 IPv4
  + Ipv4의 클래스 기반 주소 할당 방식
    - 클래스 A, B, C, D, E가 있고 클래스 D는 멀티캐스트 통신, 클래스 E는 앞으로 사용할 예비 주소로 쓰는 방식

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* + - 각 클래스는 구분 비트가 정해져 있으며 예로 클래스 A의 경우 00000000.00000000.00000000.00000000부터  
      01111111.11111111.11111111.11111111까지 주소를 할당할 수 있다.
    - 또한 네트워크의 첫번째 주소는 네트워크 주소로 사용되고 가장 마지막 주소는 브로드 캐스트용 주소로 사용된다.
    - 12.0.0.0이라는 네트워크를 부여받았다면 12.0.0.0은 네트워크 구별 주소로 사용되고 12.255.255.255는 broadcast용 주소가 되는 것
    - 이러한 방식은 사용하는 주소보다 버리는 주소가 많은 단점이 있고 이에 나온 것이 DHCP, IPv6, NAT
  + DHCP(Dynamic Host Configuration Protocol): IP 주소 및 기타 통신 매개변수를 자동으로 할당하기 위한 네트워크 관리 프로토콜. 인터넷에 접속할 때마다 자동으로 IP주소를 할당할 수 있다. (많은 라우터와 게이트웨이 장비에 DHCP 기능이 있으며 이를 통해 대부분의 가정용 네트워크에서 IP주소를 할당한다.
  + NAT(Network Address Translation) : 패킷이 라우팅 장치를 통해 전송되는 동안 패킷의 IP 주소 정보를 수정하여 다른 IP 주소로 매핑하는 것. 이를 통해 공인 IP와 사설 IP를 나눌 수 있고 많은 주소를 감당한지 못하는 IPv4의 단점을 완화

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

* + - Pros of NAT : 내부 네트워크에 대한 보안 가능, 많은 주소 감당 가능
    - Cons of NAT : 여러명이 인터넷을 접속하게 되면 호스트 숫자에 따라 접속 속도가 느려질 수 있음
* IP 주소를 이용한 위치 정보 : IP주소는 인터넷에서 사용하는 네트워크 주소, 이를 통해 동 또는 구까지 위치 추적이 가능

HTTP

* HTTP/1.0 : 기본적으로 한 연결당 하나의 요청을 처리 (하나 요청할 때마다 TCP 3 way Handshake열고 닫는다) RTT 증가한다는 단점 있음
  + RTT: 패킷이 목적지에 도달하고 나서 다시 출발지로 돌아오기까지 걸리는 시간
  + RTT 증가 해결방법
    - 이미지 스플리팅: 많은 이미지를 다운받게 되면 과부하 있기 때문에 많은 이미지가 합쳐있는 하나의 이미지를 다운받고 이를 기반으로 position을 이용하여 이미지를 표기하는 방법
    - 코드 압축: 개행 문자, 띄어쓰기 등을 없애어 코드를 압축하는 방법 (코드 용량 줄인다)
    - 이미지Base64 encoding: 이미지 파일을 64진법으로 이루어진 문자열로 인토딩하는 방법, 서버가 이미지에 대한 HTTP 요청을 따로 수행할 필요가 없어짐 but 크기가 37% 더 커지는 단점이 있음
* HTTP/1.1
  + 1.0과 다른 핵심부분은 TCP 연결을 매번 수행하는 것이 아니라 keep-alive option을 이용하여 여러 요청을 한번의 연결 안에 처리할 수 있도록 하는 것
  + HOL Blocking: 여러 개의 파일을 다운 받을 때 앞에 있는 것의 다운이 늦어지면 뒤에 것들이 대기하게 되는 현상. HTTP/1.1에서는 하나의 연결에서 여러 개를 전송가능하지만 HOL blocking은 해결하지 못했다.
  + HTTP/1.1의 헤더에는 쿠키 등 많은 메타데이터가 들어 있고 압축이 되지 않아 무거웠다.
* HTTP/2
  + HTTP/2는 SPDY 프로토콜에서 파생된 HTTP/1.x보다 지연 시간을 줄이고 응답시간을 더 빠르게 할 수 있으며 멀티플렉싱, 헤더 압축, 서버 푸시, 요청의 우선순위 처리를 지원 가능했음
  + Multiplexing: 여러 개의 스트림을 사용하여 송수신(도로 여러 개 뚫려 있다) 스트림 내의 데이터도 쪼개져 있다. 특정 스트림의 패킷이 손실되어도 해당 스트림에만 영향을 미치고 나머지 스트림은 멀쩡히 동작 가능
  + 헤더 압축: 1.x 버전의 HTTP는 무거운 헤더가 문제가 되었는데 2버젼에서는 허프만 코딩등을 이용한 HPACK 압축 형식을 가져 헤더를 압축했다.
    - Huffman Coding: 문자열을 문자 단위로 쪼개 빈도수를 세어 빈도가 높은 정보는 적은 비트 수를 사용하여 표현, 빈도가 낮은 정보는 비트 수를 많이 사용하여 표현하여 비트양을 줄이는 코딩
  + 서버 푸시: 1.1에서는 클라이언트가 서버에 요청을 해야 파일을 다운로드 받을 수 있었지만 HTTP/2는 클라이언트 요청 없이 서버가 바로 리소스 푸시 가능 (html 요청, html 전송, css 요청, css 전송이 아니라 html 요청, html 전송, css 전송 이 가능해진 것)
* HTTPS
  + HTTP/2는 HTTPS 위에서 동작, HTTPS는 애플리케이션 계층과 전송 계층 사이에 신뢰 계층인 SSL/TLS 계층을 넣은 신뢰할 수 있는 HTTP 요청을 말한다.
  + SSL/TLS : Secure Socket Layer는 여러 버전을 거쳐 TLS 1.3 (혹은 그 이상)R지 올라갔음. 즉 마지막으로는 TLS(Transport Layer Security Protocol)로 명칭이 변경되었으나 보통 SSL/TLS로 많이 부름
    - SSL/TLS은 전송 계층에서 보안을 제공하는 프로토콜. 제 3자가 메시지를 도청하거나 변조하지 못하도록 한다. (네트워크상의 인터셉팅 방지)
    - 보안 세션을 기반으로 데이터를 암호화, 보안 세션이 만들어 질 때 인증 메커니즘, 키 교환 암호화 알고리즘, 해싱 알고리즘이 사용된다.
    - 클라이언트와 서버가 키를 공유하고 이를 기반으로 인증, 인증 확인 등의 작업이 일어나는 단 한 번의 1-RTT가 생긴 후 데이터를 송수신한다.
  + 암호화 알고리즘으로는 대수곡선 기반의 ECDHE 또는 모듈식 기반의 DHE를 사용하는 것이 일반적인데 둘 모두 디피-헬만 방식을 근간으로 만들어졌다.
    - Diffie-Hellman key exchange

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

* + - * 식에서 g와 y와 p만 안다면 a 혹은 b를 알기 어렵지만 a혹은 b와 p를 안다면 y를 구하기 쉽다는 원리에 기반한 알고리즘
      * 각자 처음에 공개값을 공유하고 각자의 비밀값과 혼합한 후 혼합 값을 공유, 그 다음 각자의 비밀값과 또 혼합하면 공통의 암호키가 생성된다.
  + 해싱 알고리즘: 데이터를 추정하기 힘든 더 작고 섞여 있는 조각으로 만드는 알고리즘
    - SHA-256 알고리즘 : 해시 함수의 결괏값이 256비트인 알고리즘이며 비트코인을 비롯한 많은 블록체인 시스템에서도 쓴다.
    - 해시: 다양한 길이를 가진 데이터를 고정된 길이를 가진 데이터로 매핑한 값
    - 해싱: 임의의 데이터를 해시로 바꿔주는 일. 해시 함수가 이를 담당
  + SEO에도 도움이 되는 HTTPS
    - Search Engine Optimization : 검색 엔진 최적화를 뜻하며 구글은 SSL 인증서를 강조해왔고 HTTPS를 서비스하는 사이트가 그렇지 않은 사이트보다 SEO 순위가 높을 것이라고 공식적으로 밝힌 바 있음 (상단 노출 가능성 높다)
  + HTTPS 구축 방법
    - 직접 CA에서 구매한 인증키를 기반으로 HTTPS 서비스 구축
    - 서버 앞단의 HTTPS를 제공하는 로드밸런서를 둔다
    - 서버 앞단에 HTTPS를 제공하는 CDN을 두어 구축한다.
* HTTP/3
  + TCP위에서 돌아가는 HTTP/2와 달리 QUIC이라는 계층 위에서 작동, UDP 기반으로 돌아간다.
  + 번거로운 3-way Handshake과정 거치지 않고 1-RTT만으로 통신 시작 가능

Section 03

Operating System

운영체제: 사용자가 컴퓨터를 쉽게 다룰 수 있도록 해주는 인터페이스. 한정된 메모리나 시스템 자원을 효율적으로 분배한다.

Cf> 펌웨어: 운영체제와 유사하지만 소프트웨어를 추가로 설치할 수 없는 것

운영체제의 역할

* CPU Scheduling, Process Management: CPU 자원 특정 프로세스에 할당, 프로세스의 생성과 삭제, 자원 할당 및 반환 관리
* 메모리 관리: 한정된 메모리를 프로세스에게 할당, 관리
* 디스크 파일 관리: 디스크 파일을 어떠한 방법으로 보관할지 관리
* I/O device 관리: 마우스, 키보드 등 입력장치와 컴퓨터 간에 데이터를 주고받는 것을 관리

운영체제의 구조

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* 시스템 콜: 운영체제가 커널에 접근하기 위한 인터페이스
  + 예시 흐름
    - 유저 프로그램이 I/O 요청으로 트랩 발동
    - 올바른 요청인지 확인 후 유저모드가 시스템콜을 통해 커널 모드로 변환되어 실행

컴퓨터의 요소

* CPU
  + 산술논리연산장치, 제어장치, 레지스터로 구성되어 있는 컴퓨터 장치
  + 인터럽트에 의해 단순히 메모리에 존재하는 명령어를 해석해서 실행하는 일꾼
  + 제어장치(Control Unit aka CU): 프로세스 조작을 지시하는 CPU의 부품 입출력 장치 간 통신을 제어하고 데이터 처리 순서를 결정
  + 레지스터: CPU 안에 있는 매우 빠른 임시기억장치. CPU는 자체적으로 데이터를 저장할 방법이 없기 때문에 레지스터를 거쳐 데이터를 전달
  + 산술논리연산장치(ALU: Arithmetic Logic Unit): 논리합, 논리곱 등 논리 연산을 계산하는 디지털 회로
  + CPU의 연산 처리 예시
    - 제어장치가 메모리에 계산할 값 로드, 레지스터에 로드
    - 제어장치가 레지스터에 있는 값을 계산하라고 ALU에게 명령
    - 제어장치가 계산된 값을 레지스터에서 메모리로 저장
  + Interupt: 어떤 신호가 들어왔을 때 CPU를 잠깐 정지시키는 것을 말한다. IO 디바이스로 인한 인터럽트, divide 0 Interupt 등등으로 발생
    - 박찬호 CPU 예시 https://whatisthenext.tistory.com/147
    - 인터럽트가 발생되면 인터럽트 핸들러 함수가 모여 있는 인터럽트 벡터로 가서 인터럽트 핸들러 함수가 실행된다.
    - 인터럽트 간에는 우선순위가 있고, 우선순위에 따라 실행되며 인터럽트는 하드웨어 인터럽트, 소프트웨어 인터럽트 두 가지로 나뉜다.
      * 하드웨어 인터럽트: 디바이스에서 발생하는 인터럽트. 원하는 디바이스로 향해 디바이스에 있는 작은 로컬 버퍼에 접근하여 일을 수행
      * 소프트웨어 인터럽트 (aka trap): 프로세스 오류 등으로 프로세스가 시스템콜을 호출할 때 발동
* DMA Controller
  + DMA 컨트롤러는 I/O 디바이스가 메모리에 직접 접근할 수 있도록 하는 하드웨어 장치 CPU의 부하를 막아주는 보조 일꾼
* Memory
  + 메모리는 전자회로에서 데이터나 상태, 명령어 등을 기록하는 장치를 말하며, 보통 RAM을 일컬어 메모리라고도 한다.
* Timer
  + 몇 초 안에는 작업이 끝나야 한다는 것을 정하고 특정 프로그램에 시간제한을 다는 역할을 한다.
* Device Controller
  + 컴퓨터와 연결되어 있는 IO 디바이스들의 작은 CPU를 말한다.