융합프로젝트

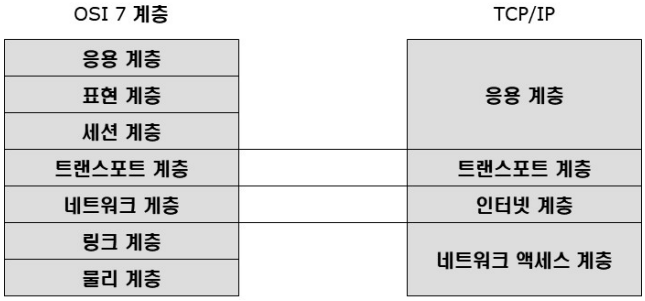
20150228 김시헌

Chapter 1 네트워크 개요

▣ 컴퓨터 통신 프로토콜

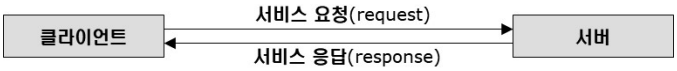
▷ OSI 7계층 구조

- Internet은 TCP/IP 통신 프로토콜 기반으로 동작



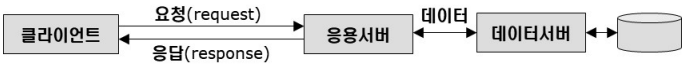
▣ 네트워크 프로그램 구현 모델

▷ 2-tier Client-Sever model



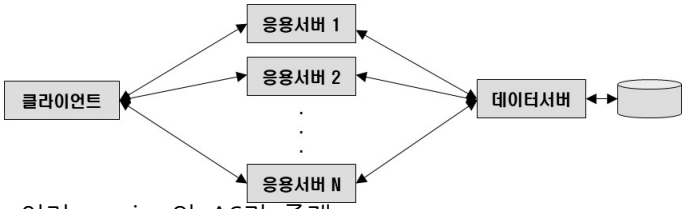
- 대부분의 통신 Program
- Server에서의 병목 현상
- C의 증가는 S에 Traffic 집중과 처리 용량 부족 현상 발생
- Fat Client : S의 기능을 C에 구현. Program의 업그레이드, Version 관리 문제 발생
- Thin Client - Fat Server : 대다수, 서버 기능 多

▷ 3-tier Client-Sever model



- S를 AS, DS로 구분
- C는 AS에 request, AS는 DS에 D를 얻어 C에 response
- C는 DS의 정보가 필요 없음
- C가 같은 요청을 동시에 하면 AS는 1번만 처리

▷ n-tier Client-Sever model



- 여러 version의 AS가 존재
- C가 필요에 따라 다른 AS를 선택할 수 있음
- Service 제공 도중 새로운 AS 추가 가능

▣ 서버 구현 기술

▷ 연결형과 비 연결형 서버

항목	연결형	비연결형
연결방식	• TCP: 연결 기반 (Connection-Oriented) • 연결 후 통신 (각 클라이언트마다 연결) • 1:1 통신 방식 (유니캐스트용) • 클라이언트 수가 증가하면 서버 부담 큼	• UDP: 비연결 기반 (Connectionless-Oriented) • 연결 없이 통신 (클라이언트마다 연결 X) • 1:1, 1:n 통신 방식 (방송, 멀티캐스트용) • 클라이언트 수가 증가해도 서버 부담 적음
특징	• 데이터의 경계 구분 안함 (Byte Stream) • 신뢰성 있는 데이터 전송 • 데이터의 전송 순서 보장 • 데이터의 수신 여부 확인 (손실되면 재전송) • 패킷을 관리할 필요 없음 • UDP 보다 전송 속도 느림	• 데이터의 경계 구분함 (Datagram) • 신뢰성 없는 데이터 전송 • 데이터의 전송 순서 보장 못함 • 데이터의 수신 여부 확인 안함 (손실되어도 알 수 없음) • 패킷을 관리해야 함 • TCP 보다 전송 속도 빠름

- 근거는 신뢰성!
- Unicast 1:1, broadcast + multicast 1:n
- 카카오톡은 Unicast가 여러번 이루어진다.
- Data 구분 : TCP 데이터 단위가 달라질 수 있다(Buffer에서 합치고 분리함)

▷ Stateless와 Stateful 서버

- Stateful Server : S가 C의 State 유지
- Stateless Server : S가 C의 State 유지 X
- network가 안정적일 경우 Stateful 이 유리함. 그러나 Internet 환경은 Stateless Server를 사용하는 것이 안전

▷ Iterative Server (반복)

- C의 요청을 순서대로 처리
- C의 요청이 짧은 시간에 처리할 수 있는 경우 적합
- Concurrent Server보다 구현이 간단함

▷ Concurrent Server (동시)

- C의 요청을 동시에 처리
- 다중 처리 기능이 필요
- 이상적인 서버의 기능으로는 가능한 많은 C가 접속해야 하고, S는 각 C의 request에 빠른 response를 보이며 고속으로 Data를 전송해야 함. 또한, System 자원 사용을 최소화해야 함

Chapter 2 Java IO Stream

▣ Stream

▷ 특징

- Java는 OS와 System에 관계 없이 동작하도록 키보드, 파일, network 및 단말기 등과 같은 모든 IO 장치에 Stream을 이용하여 입/출력 수행
- 순서가 있는 일련의 데이터의 흐름을 의미

▷ I/O Stream 특징

- 기본 단위 : 8bit (byte) 길이의 이진수
- 단방향 Stream, FIFO 구조를 띰

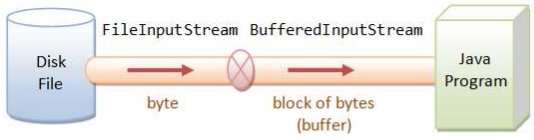
▷ Byte(8 bits) I/O Stream

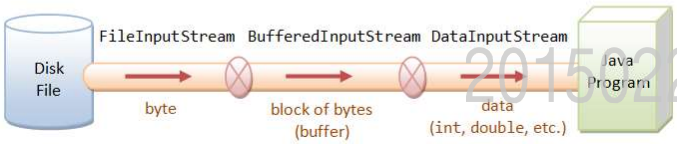
- byte 흐름으로 처리. binary file을 읽는 입력 Stream

▷ Byte I/O의 문제점
- 1byte 단위로 Read, Write
- byte단위이므로 문자 표현에 적합하지 않음(한글 등)
- 문자 I/O Stream을 이용하여 해결함

▷ 문자(16 bits) I/O Stream
- 알파벳, 한글 등 문자의 흐름으로 처리(16 bits 길이의 Unicode 사용)
- 문자가 아닌 Binary Data는 처리 불가
- Read : byte data를 문자로 변환하여 Read
- Write : 문자를 byte Data로 변환(Incoding) 후 전송
- 한 글자씩 Read, Write (16bits 길이의 Unicode 문자)
- 한 글자씩 Read, Write하면 속도가 매우 느림
- **Buffer를 사용하여 한 줄씩 Read, Write 하여 해결**

▷ I/O Stream Chain
- byte를 buffer로 변환하고 필요에 따라서 자료형까지 변환하는 과정





Chapter 3 소켓 기초

▣ Socket

▷ Socket
- AL과 TL 間 Interface
- UNIX의 Socket은 매우 상세하게 제공됨
- Window와 Java에선 User가 편하게 사용
- Socket은 특정 Port #과 연결되어 있다
- **즉, Port #은 Socket을 구분한다. = 전달할 process를 구분**

▷ TCP에서의 Socket
- Source IP, Source Port #, Dest IP, Dest Port #
- ServerSocket : TCP Server에서 사용되는 Package
- Socket : Server와 Client에서 사용

▷ UDP 에서의 Socket
- DatagramSocket : Datagram의 송/수신에서 사용(S,C)
- DatagramPacket : UDP에서 사용되는 Datagram(Format)

▣ 인터넷 주소

▷ IP 주소(IPv4 : 32bit, IPv6 : 128bit)
- IP Datagram을 Dest host까지 전달할 때 사용
- 특정 host를 찾을 때 사용

▷ Port # (16bit)
- Data를 최종적으로 전달할 process 구분
- host내의 통신 socket을 구분할 때 사용
- 같은 port #를 TCP와 UDP가 동시에 사용 가능

▷ Well-known port
- 1023번 이하가 배정되어 사용됨
- 널리 사용되는 service를 위해 미리 지정된 port #
- FTP(21, 22), mail(25), http(80) 등등

▷ InetAddress Class
- IP 주소로부터 Domain Name을 반환(DNS)
- Host name과 IP 주소를 저장하는 field 포함
- byte getAddress(), String getHostAddress() : 현재의 IP 반환
- getByName(String hostName) : 이름을 통해서 InetAddress 객체 반환 - 첫 번째 호스트 객체를 반환

▣ 주의 사항

▷ Program 間 통신
- 다른 언어 사용 時 주의해야 할 사항으로는

1. Byte 순서 (가장 치명적)
2. 고유 형식
3. 자료형 길이 등이 있다.

▷ byte 순서
- 2byte 이상으로 구성된 정보(port #, IP 주소)를 byte 단위로 전송하는 순서
- Big-endian : 큰 단위가 앞으로
- Little-endian : 작은 단위가 앞으로
- 모두 Big-endian으로 저장해야 한다

▷ 고유 형식(Java Object Implements)
- 통신을 위해 병렬적 Data를 직렬화한다.
- 언어에 따라서 Format이 다르다.

▷ 자료형 길이
- Java는 고정되어있고 C에서는 고정 X

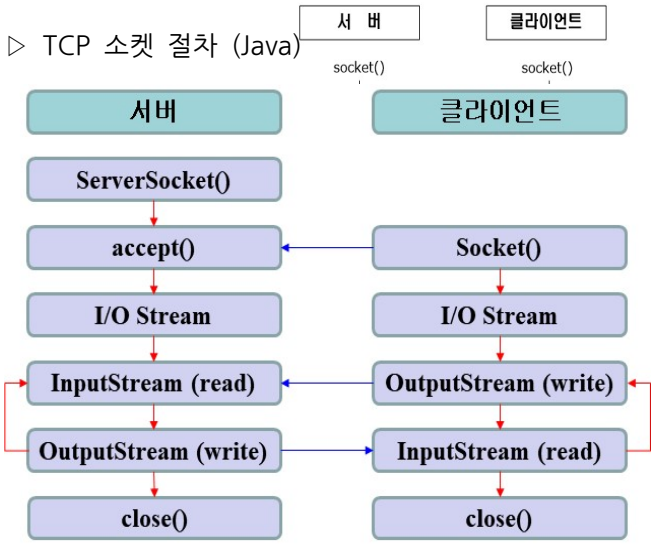
▣ TCP Socket

▷ TCP 소켓 절차 (C)

-bind() : port # 묶음

-listen() : request 대기

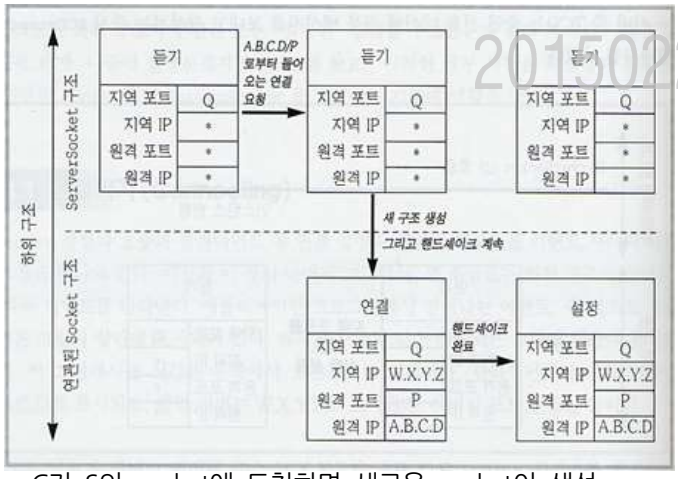
▷ TCP 소켓 절차 (Java)



- 사용자 편의성을 위해 ServerSocket()에서 내부적으로 bind()와 listen() 통합

▷ Socket으로부터 Stream 객체 얻기 / Data 송수신
- Socket 객체 생성만으로는 S와 통신 불가
- 통신을 위해 App과 Socket을 연결하는 IO Stream 생성 필요
- IO Stream의 read(), write() method 사용
- network로부터 Data 입출력은 byte 및 문자 Stream 이용

▷ 접속 요청받기 및 S의 socket 생성



- C가 S의 socket에 도착하면 새로운 socket이 생성
- C가 엄청 많다면 S socket이 터지므로 접속만 담당하는 S 측 Socket (main)이 있음. 그 후 통신은 새 socket과 함
- 새로운 Socket들의 Port #는 다 똑같음. 그래서 send의 IP와 Port #이 필요함
- 아무튼, ServerSocket은 bind()랑 listen()만 하는 놈
- 그러니까 ServerSocket은 close()되면 안됨.

▷ close()의 문제점
- close()는 호출 후 데이터 송수신이 불가능함
- S나 C나 호출 가능함
- 호출한 시점에서의 송신 buffer는 3가지 옵션이 있음.

1. 아직 전송되지 못한 데이터를 다 전달 후 종료
2. 전달 중인 데이터만 모두 전달 후 종료
3. 미전송 데이터를 모두 버리고 종료

- S의 입장에서 C가 Data를 다 읽었다고 확신 X
- S가 추가 Message를 전송할 때 C가 수신 불가
- 이에 대한 해결책
- shutdownInput()과 shutdownOutput() 사용
- 이는 송수신만 막음. memory 할당은 되어있는 상태, 즉 Zombie 상태로 변함. 끝나고 close() 해줘야 함
- 더 수신할 데이터가 없고 송신할 데이터도 없다면 그때 close()를 사용한다. (read의 반환 값이 -1이거나 NULL일 때)

▷ UDP
- DatagramSocket() : Datagram Socket 생성
- DatagramPacket(byte[] buffer, int length) : 목적지가 없으므로 수신용으로 사용
- DatagramPacket(byte[] buffer, int length, InetAddress remoteAddr, int remotePort) : 송신용으로 사용
- Socket으로부터 입출력 Stream Object를 받지 않음

Chapter 4 TCP 데이터 경계

▣ 데이터 경계

▷ TCP
- 데이터 경계를 구분하지 않음(Byte Stream)
- Read()는 buffer 크기만큼 Socket 수신 buf에서 data read
- App에서 Data 경계를 위한 별도의 작업이 필요함
- 길이 정보를 같이 보낸다,

▷ UDP
- 데이터 경계를 구분함(Datagram)
- Receive()는 한 번에 하나의 Data Packet을 Read
- 만약 Data_length가 buffer보다 크면 buffer만큼 읽고 나머지는 Socket 수신 buffer에서 폐기함
- App에서 Data 경계를 위한 별도의 작업 필요 없음

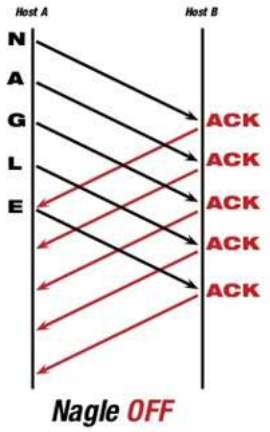
▷ TCP - 데이터 경계 구분 방법

1. **고정적인 길이**로만 보내기
2. **끝부분에 EOR**(End Of Record) 를 붙여 보냄. 수신자는 EOR이 나올 때까지 Data 수신
3. Send는 Data를 **고정 길이 + 가변 길이**로 보냄. Receive는 고정 길이 Data를 읽어서 뒤의 가변 Data 길이를 알아냄. 그리고 그만큼만 수신(권장방법)
4. Send는 가변 길이 데이터 전송 후 Socket을 종료. Receive는 read()의 **반환 값이 -1이 될 때까지 Data 수신**

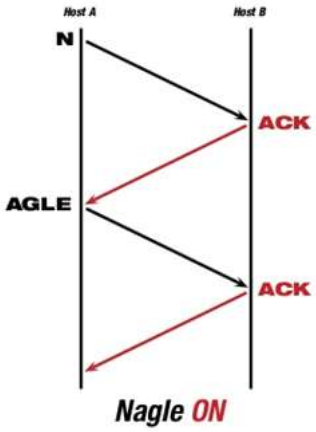
Chapter 5 Socket Option

▣ 데이터 경계

▷ Nagle Algorithm
- Ack를 받기 전까지는 작은 Data는 전송하지 않고 대기
- 효율적이지만 Delay 증가



Nagle OFF



Nagle ON

▷ SO_KEEPALIVE
- 일정 시간(2시간) 동안 통신이 없으면 연결 유지를 확인하는 packet(keep alive prove)을 전송함
- keep alive prove의 3가지 응답

1. ACK : TCP 연결 정상 동작
2. RST : 상대 host가 꺼진 후 재부팅된 상태이므로 종료
3. 응답 없음 : 여러 번 보내보고 12분간 응답 없으면 종료

▷ SO_REUSEADDR
- FIN packet을 보내고 ACK가 오면 Half-Close, ACK 후 연속으로 FIN packet이 오면 ACK를 보내고 2MSL (Maximum Segment Lifetime)을 기다리고 종료.
- FIN 보내고 ACK를 받아도 바로 종료하지는 않는 방식. 재사용을 허용한다.
- Server 쪽에 꼭 있어야 한다.

▷ SocketAddress
- 기본 생성자를 제외하고 아무런 Method 제공 X 추상.

Chapter 6 다중처리 기술

▣ TCP Server-Client의 문제점

▷ 문제 1
- 동시에 둘 이상의 Client Service 불가
- 송신과 수신을 교대로 진행
- 해결방안

1. 짧은 통신 : 각 C와 통신하는 시간을 짧게 함. 구현이 쉽고 시스템 자원 적게 사용. 그러나 각 C의 처리 지연 시간이 길어질 수 있음.
2. 멀티 프로세스 : 송신과 수신, 그리고 각 C를 Process를 이용해 독립적으로 처리. 구현이 쉽지만 가장 많은 시스템 자원을 사용함.
3. 멀티 쓰레드 : 송신과 수신, 그리고 각 C를 Thread를 이용하여 독립적으로 처리. 구현이 쉽지만 많은 시스템 자원을 사용함.
4. Non Blocking IO : 입출력 대상을 묶어서 관리하는 방식으로 Service 제공. 소수의 Thread를 이용해 다수의 C를 처리. 구현이 어려움

▷ Non-Blocking IO
- Blocking : read() 호출한 경우 수신된 Data가 있으면 return. 만약 없다면 기다림(Blocking)
- 한 Process(Thread) 내에서 이루어지는 다중 처리 방법

1. C : IOCP, EPOLL
2. Java : Selector

Chapter 7 Thread

▣ 자바 Thread와 JVM

▷ Java Thread
- JVM에 의해 Schedule되는 실행 단위의 Code Block
- Thread의 생명 주기는 JVM이 관리
- JVM은 Thread 단위로 Scheduling

▷ JVM과 MultiThread의 관계
- 하나의 JVM은 하나의 JAVA APP만 실행
- JAVA APP이 시작될 때 JVM도 함께 실행됨
- JAVA APP이 종료될 때 JVM도 함께 종료됨
- MultiProcess 지원 안함
- 하나의 APP은 하나 이상의 Thread로 구성 가능

▷ Thread 종료
- Java에선 Program이 Main 끝날 때 끝나는 게 아니라 thread가 전부 다 종료되어야 Program 종료

▣ Thread Synchronization

▷ 공유 Data
- Thread의 Stack 영역 Data 공유 불가
- Thread의 Heap과 Class 영역 Data 공유 가능

▷ Synchronization
- 여러 Thread가 공유 Data에 접근할 때 공유 Data의 값을 정확하게 예측할 수 없는 문제
- Method로 선언하는 방식과 Block 방식이 있음
- Block 방식은 Lock Object를 명시적으로 설정해야 함
- 성능 향상을 위해 Block 방식 선호

```
void 메소드명(파라미터) {
  // 여러 쓰레드 동시 실행 가능
  synchronized(this) {
    // 단 하나의 쓰레드만 실행 가능
  }
  // 여러 쓰레드 동시 실행 가능
}
```

synchronized 블록(Block)

- 기타 Thread 동기화는 ppt 53~62 참고.

Chapter DB

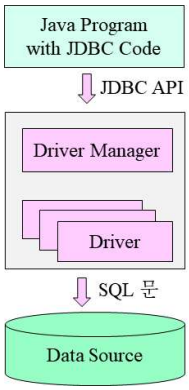
▣ SQL Programming Interface

▷ 특징

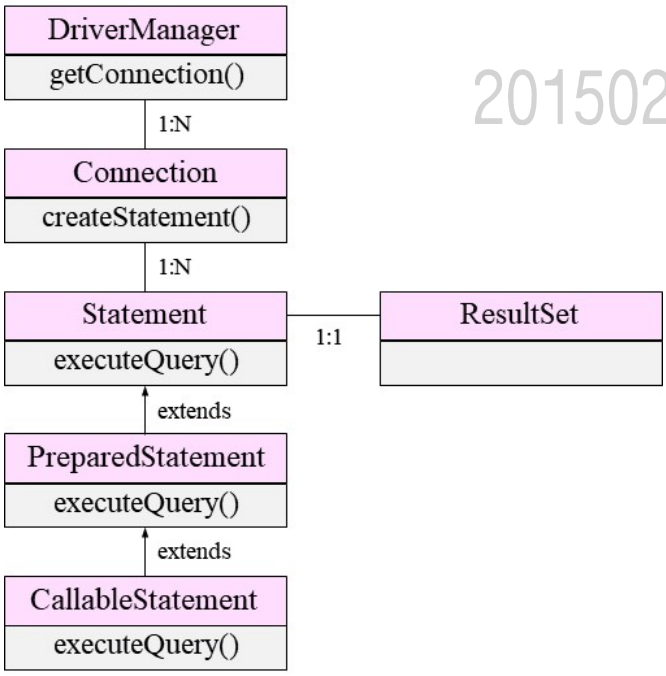
- 프로그래밍 언어에서 SQL을 사용하여 DB에 접근하는 방법

1. Stored Procedures : PL/SQL (Oracle)
2. Embedded SQL (SQLJ)
3. Call-Level Interface : JDBC (JavaSoft), ODBC (MS)

▷ 다음은 JDBC를 사용하여 DB에 접근하는 방법임



응용	JDBC 메소드 호출 • 데이터소스 접속/단절 • SQL문 전송/결과 검색 • 트랜잭션 제어
드라이버 매니저	• 특정 DBMS를 위한 드라이버를 검색/적재/초기화 • JDBC 호출이 올바른지 확인함
드라이버	SQL문 전송 및 결과 반환 • 데이터소스에 접속 • SQL 요청 번역/최적화; 데이터소스에 전송 • 결과 반환 및 커서 조작(result set 경우) • DBMS 에러코드를 JDBC 에러코드로 변환 • (명시적 요청時) 트랜잭션을 시작시킴
데이터소스	응용프로그램이 원하는 데이터를 보유



▷ DriverManager : 드라이버 관리

- getConnection() → 호출 시 접속 완료
- conn = DriverManager.getConnection(url, id, pw)

▷ Connection : 접속한 상태의 객체 (conn)

- createStatement() → 접속한 상태의 객체 생성
- Statement = connection.createStatement()

▷ Statement : 상태 Interface를 구현한 객체

- executeQuery(sql) : sql문 실행
- ResultSet : Query가 실행되어 얻어진 결과 행들의 집합
- rs = statement..executeQuery(sql)