7 TCP/IP 서버 소켓

8 TCP/IP 클라이언트 소켓

개요

□ **인터넷의 전송 계층에서의 대표적인 프로토콜의 종류**

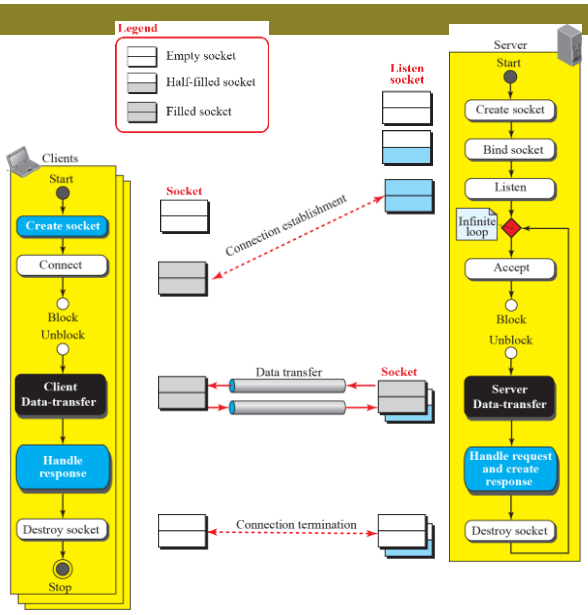
- 전송 계층에서의 역할은 포트를 사용한 두 개의 종단 호스트(end hosts)간에 데이터를 전달하는 기능이다. 대표적인 프로토콜의 종류에는 TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)가 있다.
- TCP(Transmission Control Protocol)는 두 개의 종단 간에 연결을 설정한 후에 데이터를 바이트 스트림(stream)으로 교환하는 연결형(Connection-Oriented) 프로토콜이며 신뢰성이 있다.
- UDP(User Datagram Protocol)는 두 개의 종단 간에 연결을 설정하지 않고 데이터를 교환하는 비연결형(Connectionless) 프로토콜이며 신뢰성이 없다

소켓 프로그래밍

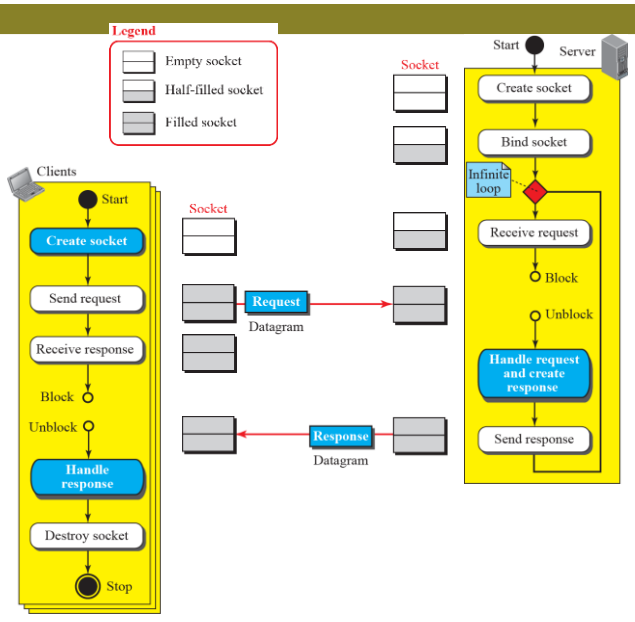
□ 소켓 (socket)

- 소켓은 네트워크 상에서 수행되는 두 프로그램 간의 양방향 통신 링크의 한쪽 끝 단을 의미
- 소켓은 특정 포트 번호와 연결되어 있음
  - 전송계층에서 데이터를 보낼 응용프로그램을 식별할 수 있음
- 소켓 종류
  - 서버 소켓과 클라이언트 소켓

TCP 통신을 위한 신호 흐름도



UDP 통신을 위한 신호 흐름도



소켓의 종류

TCP 소켓의 종류

- **ServerSocket** 클래스: 서버를 위한 소켓 연결용 소켓

서버(server) 컴퓨터

- **Socket** 클래스: 클라이언트를 위한 소켓 통신용 소켓

클라이언트(client) 컴퓨터

클라이언트와 서버 연결 순서

□ 클라이언트와 서버 연결

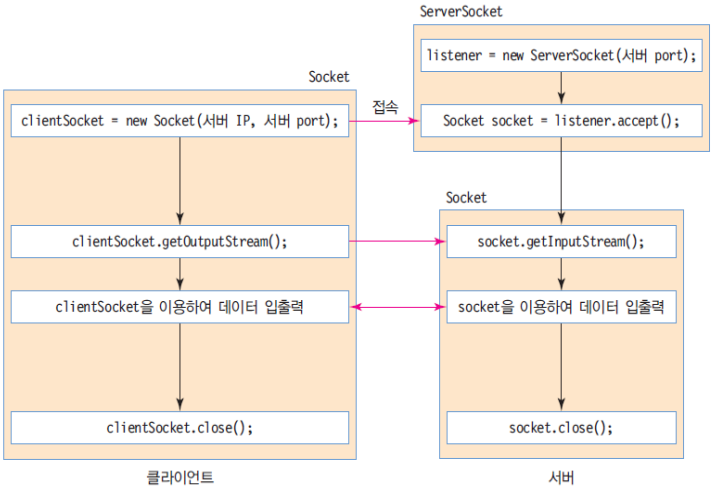
- 서버는 서버 소켓으로 들어오는 연결 요청을 기다림

- 클라이언트가 서버에게 연결 요청

- 서버가 연결 요청 수락하고 새로운 소켓을 만들어 클라이언트와 연결 생성

7/19

TCP 소켓을 이용한 서버 클라이언트 통신 프로그램의 구조



8/19

ServerSocket 클래스

ServerSocket 클래스

- 서버 소켓에 사용되는 클래스
- java.net 패키지에 포함
- 생성자

생성자	설명
public ServerSocket(int port) throws IOException	포트 번호 port에 대해 ServerSocket의 새로운 인스턴스를 만든다. 포트 번호 0는 비어있는 포트 번호를 사용한다는 의미이다. queue는 서버가 받을 수 있는 입력 연결의 개수를 의미한다.(디폴트는 50 연결이다.) addr는 컴퓨터의 인터넷 주소를 나타낸다.
public ServerSocket(int port, int queue)	
public ServerSocket(int port, int queue, InetAddress addr)	

ServerSocket 클래스

- 메소드

메소드	설명
public Socket accept()	접속 요청을 받는다.
public void close()	ServerSocket을 닫는다.
public Inetaddress getInetAddress()	소켓이 연결되어 있는 인터넷 주소를 반환한다.
public int getSoTimeout()	소켓에 대한 타임아웃 값을 밀리 초로 반환
public int getLocalPort()	서버 소켓이 연결 요청을 모니터링하는 포트 번호 반환
public boolean isBound()	서버 소켓이 로컬 주소에 연결되어있으면 true 반환
public boolean isClosed()	서버 소켓이 닫혀있으면 true 반환

Socket 클래스

☐ Socket 클래스

- 클라이언트 소켓에 사용되는 클래스
- java.net 패키지에 포함
- 생성자

생성자	설명
Socket(String host, int port)	호스트 이름이 host이고 포트 번호가 port인 새로운 소켓을 생성한다.
Socket(InetAddress address, int port)	InetAddress에 기술된 주소로 새로운 소켓을 생성한다.

11/19

Socket 클래스

- 주요 메소드

메소드	설명
InputStream getInputStream()	소켓이 사용하는 입력 스트림을 반환한다.
OutputStream getOutputStream()	소켓이 사용하는 출력 스트림을 반환한다.
Inetaddress getInetAddress()	소켓이 연결되어 있는 인터넷 주소를 반환한다.
public int getLocalPort()	소켓이 연결되어 있는 포트 번호를 반환한다.
public int getPort()	원격 컴퓨터의 포트 번호를 반환한다.
public InetAddress getLocalAddress()	소켓이 연결되어 있는 인터넷 주소를 반환한다.

void close()	소켓을 닫는다.
void connect(SocketAddress endpoint)	소켓을 서버에 연결
boolean isBound()	소켓이 로컬 주소에 연결되어있으면 true 반환
boolean isConnected()	소켓이 서버에 연결되어 있으면 true 반환
boolean isClosed()	소켓이 닫혀있으면 true 반환
void setSoTimeout(int timeout)	데이터 읽기 타임아웃 시간 지정. 0이면 타임아웃 해제.

12/19

서버 소켓 생성, 클라이언트 접속, 입출력 스트림 생성

□ 서버 소켓 생성

```
ServerSocket serverSocket = new ServerSocket(5550);
```

- 이미 사용 중인 포트 번호를 지정하면 예외가 발생

□ 클라이언트로부터 접속 기다림

```
Socket socket = serverSocket.accept();
```

- accept() 메소드는 연결 요청이 오면 새로운 Socket 객체 반환
- 서버에서 클라이언트와의 데이터 통신은 새로 만들어진 Socket 객체를 통해서 이루어짐
- ServerSocket 클래스는 Socket 클래스와 달리 주어진 연결에 대해 입출력 스트림을 만들어주는 메소드가 없음

□ 네트워크 입출력 스트림 생성

```
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
```

- accept() 메소드에서 얻은 Socket 객체의 getInputStream()과 getOutputStream() 메소드를 이용하여 데이터 스트림 생성
- 일반 스트림을 입출력하는 방식과 동일하게 네트워크 데이터 입출력

클라이언트 소켓 생성, 서버 접속, 입출력 스트림 생성

□ 클라이언트 소켓 생성 및 서버에 접속

```
Socket clientSocket = new Socket("128.12.1.1", 5550);
```

- Socket 객체의 생성되면 곧 바로 128.12.1.1의 주소로 자동 접속

□ 네트워크 입출력 스트림 생성

```
BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
BufferedWriter out = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));
```

- 일반 스트림을 입출력 하는 방식과 동일

□ 서버로 데이터 전송

```
out.write("hello"+"\n");
out.flush();
```

- flush()를 호출하면 스트림 속에 데이터를 남기지 않고 모두 전송

□ 서버로부터 데이터 수신

```
int x = in.read();       // 서버로부터 한 개의 문자 수신
String line = in.readline(); //서버로부터 한 행의 문자열 수신
```

□ 네트워크 접속 종료

```
clientSocket.close();
```

클라이언트로 데이터 송수신

□ 클라이언트로부터 데이터 수신

```
int x = in.read();              // 클라이언트로부터 한 개의 문자 수신
String line = in.readline(); //클라이언트로부터 한 행의 문자열 수신
```

□ 클라이언트로 데이터 전송

```
out.write("Hi!, Client"+"\n");
out.flush();
```

- flush()를 호출하면 스트림 속에 데이터를 남기지 말고 모두 전송

□ 네트워크 접속 종료

```
socket.close();
```

□ 서버 응용프로그램 종료

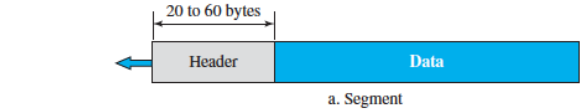
```
serverSocket.close();
```

- 더 이상 클라이언트의 접속을 받지 않고 서버 응용 프로그램을 종료하고자 하는 경우 ServerSocket 종료

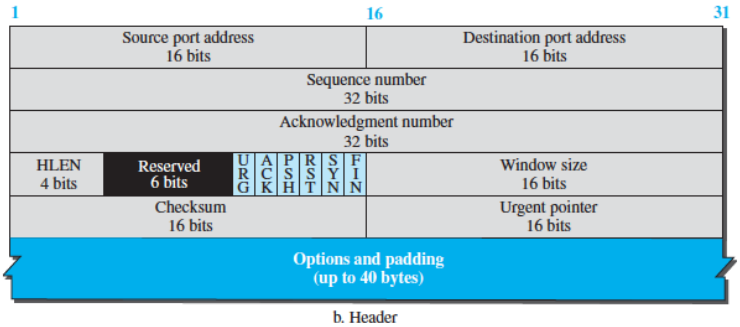
15/19

2절. TCP

□TCP 헤더



a. Segment



b. Header

16/19

SO_KEEPALIVE 옵션

□ 용도

- TCP 프로토콜 수준에서 연결 여부를 확인하기 위해 상대 TCP에 주기적으로(약 2시간 간격) Keep Alive Probe 패킷을 보냄
- Keep Alive Probe에 대한 세가지 응답
  - 상대방이 ACK를 보내옴
  - 상대방이 RST를 보내옴
  - 아무 응답이 없음
    - 몇 번 더 시도한 후 연결을 종료

SO_REUSEADDR 옵션

□ 용도

- 사용 중인 포트 번호를 재사용
  - 사용 중인 IP 주소와 포트 번호로 bind() 함수를 (성공적으로) 호출할 수 있음

□ 목적

① 서버 종료 후 재실행시 bind() 함수에서 오류가 발생하는 것을 방지

② 두 개 이상의 IP 주소를 가진 호스트에서 각 IP 주소별로 서버를 따로 운용

③ 멀티캐스팅 응용이 동일한 포트 번호를 사용할 수 있도록 함 (완전중복 바인딩)

TCP_NODELAY 옵션

- 용도
  - Nagle 알고리즘 작동 여부 결정
- Nagle 알고리즘
  - ① 보낼 데이터가 MSS(maximum segment size)로 정의된 크기만큼 쌓이면, 상대편에게 무조건 보냄
  - ② 보낼 데이터가 MSS보다 작을 경우, 이전에 보낸 데이터에 대한 ACK가 오기를 기다림. ACK가 도달하면 보낼 데이터가 MSS보다 작더라도 상대에게 보냄
- **Nagle 알고리즘의 장단점**
  - ➢ 장점: 작은 패킷이 불필요하게 많이 생성되는 것을 미연에 방지함으로써 네트워크 트래픽을 감소시킴
  - ➢ 단점: 데이터가 충분히 쌓일 때까지 또는 ACK가 도달할 때까지 대기하는 시간 때문에 응용의 반응 시간이 길어질 가능성이 있음