UDP 기반
서버 클라이언트

인하공업전문대학 컴퓨터정보과
최효현 교수

주요사항

- UDP에 대한 이해
- UDP 기반 에코 서버/클라이언트의 구현
- UDP의 데이터 경계 (Boundary)
- connect()를 이용한 UDP 프로그래밍

6.1 UDP에 대한 이해

- **UDP 소켓의 특성**

  - **UDP(User Datagram Protocol)**
  - **IP**를 기반으로 데이터를 전송한다.
  - 흐름제어**(flow control)**을 하지 않기 때문에 데이터 전송을 보장 받지 못함

    => 신뢰할 수 없는 데이터 전송
  - 신뢰성이 없는 메시지 전송 : 순서번호도 사용않음
  - 연결설정 및 연결 종료 과정도 존재 않음.
  - 연결상태가 존재하지 않음
  - 프로토콜 자체가 상당히 간단
  - **TCP** 방식보다 속도가 빠름

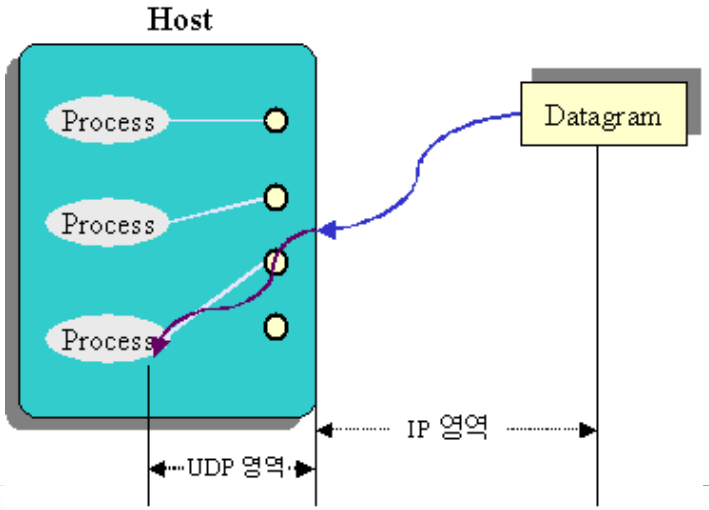
6.1 UDP에 대한 이해

- **UDP의 내부 동작**

1. 기본 데이터 단위 **:** 원래는 “세그먼트”이나 **TCP**에서만 “세그먼트”라 하고, **UDP**에서는 “데이타그램” 이라고 더 많이 칭함

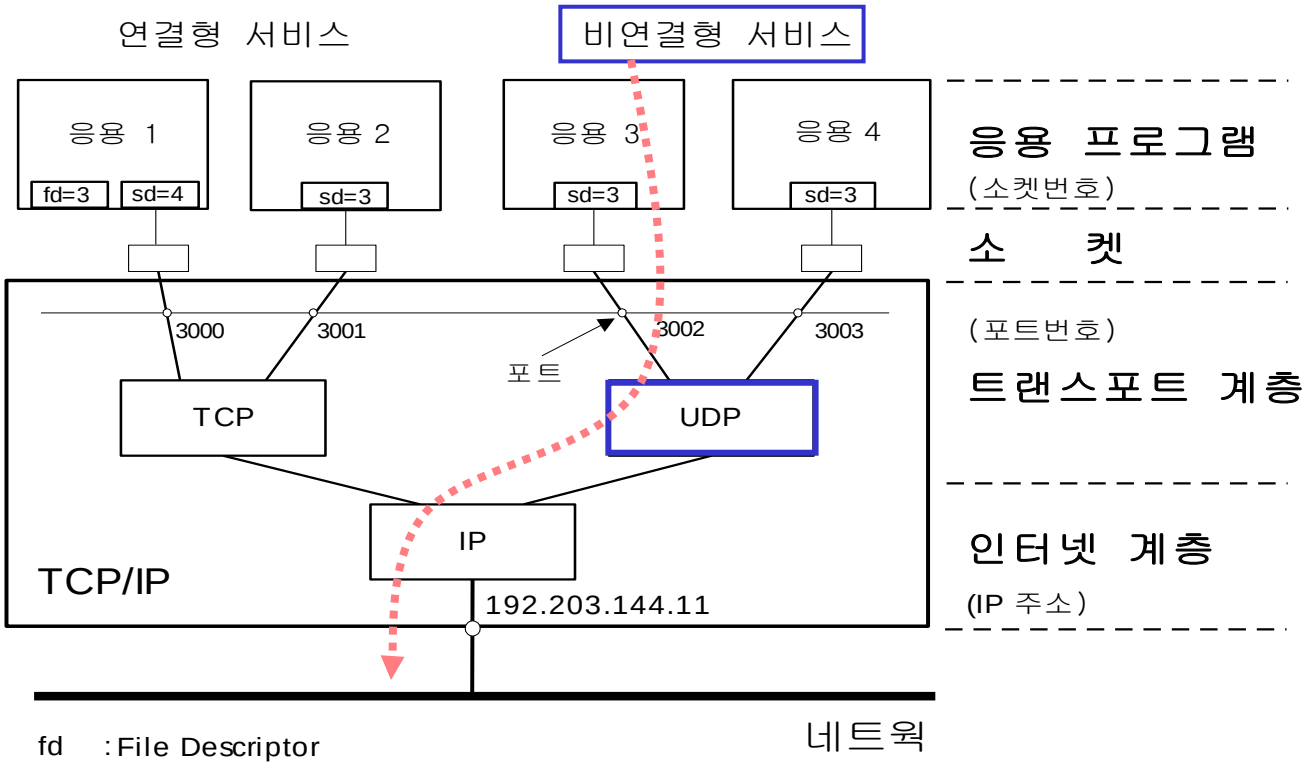
2. IP(L3 프로토콜**)**가 데이터를 라우팅을 통해 목적지 까지 전달

3. UDP(L4 프로토콜)는 호스트내에서 **Port**를 통해 최종 목적지 프로세스를 구별



6.1 UDP에 대한 이해

- **UDP의 내부 동작**



fd : File Descriptor
IP : Internet Protocol
sd : Socket Descriptor
TCP : Transmission Control Protocol
UDP : User Datagram Protocol

6.1 UDP에 대한 이해

- **UDP의 효율적 사용**

  1. UDP도 그런대로 신뢰할 만함

  => 응용에 따라 TCP나 UDP를 선택

  예) 압축화일 : TCP가 유리

  실시간서비스(영상 및 음성) : UDP가 유리

  2. 항상 UDP가 TCP보다 빠른것은 아님

  => TCP가 느린이유 : 연결설정 및 종료절차, 흐름제어

  => 데이터 양이 많은 경우(또는 세션이 긴경우) : TCP 가 유리

  데이터의 양이 적은 경우 : UDP가 빠름

6.2 UDP기반 서버/클라이언트 구현

- **UDP서버는 클라이언트와 연결되어 있지 않음**

  - 서버와 클라이언트간에는 연결설정**(Connection)** 상태가 존재 않음
  - 일반적으로 연결 설정과정을 거치지 않는다.

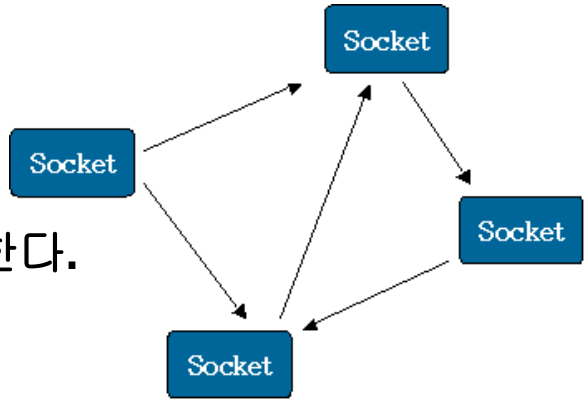
    => TCP에서 필요하던**, listen(), accept(), connect()** 함수가 불필요
  - **UDP** 기반 서버와 클라이언트는 소켓설정과 주소할당 만이 필요

- **UDP의 소켓은 오직 하나임**

  - 데이터를 주고 받기 위한 소켓은 하나만 생성한다.

    => TCP에서는 서버에 **10**개의 클라이언트가 연결

    : 소켓 **10**개 생성
  - 소켓 하나로 여러 개의 클라이언트와 송수신 가능



6.2 UDP기반 서버/클라이언트 구현

- **UDP기반의 데이터 입출력 함수**
  - 데이타 전송함수

```
int sendto(int sock, const void* msg, int len, unsigned flags,
           const struct sockaddr * addr, int addrlen)
```

성공시 : 전송한 바이트수
실패시 : -1 리턴

sock: 소켓 디스크립터를 인자로 넘김
msg : 전송할 데이터의 버퍼 포인터
len : 길이
flags : 0로 설정, 잘 사용되지 않음
addr : 전송 하고자 하는곳의 주소정보
addrlen : 주소 구조체 변수길이

6.2 UDP기반 서버/클라이언트 구현

- **UDP기반의 데이터 입출력 함수**
  - 데이타 수신함수

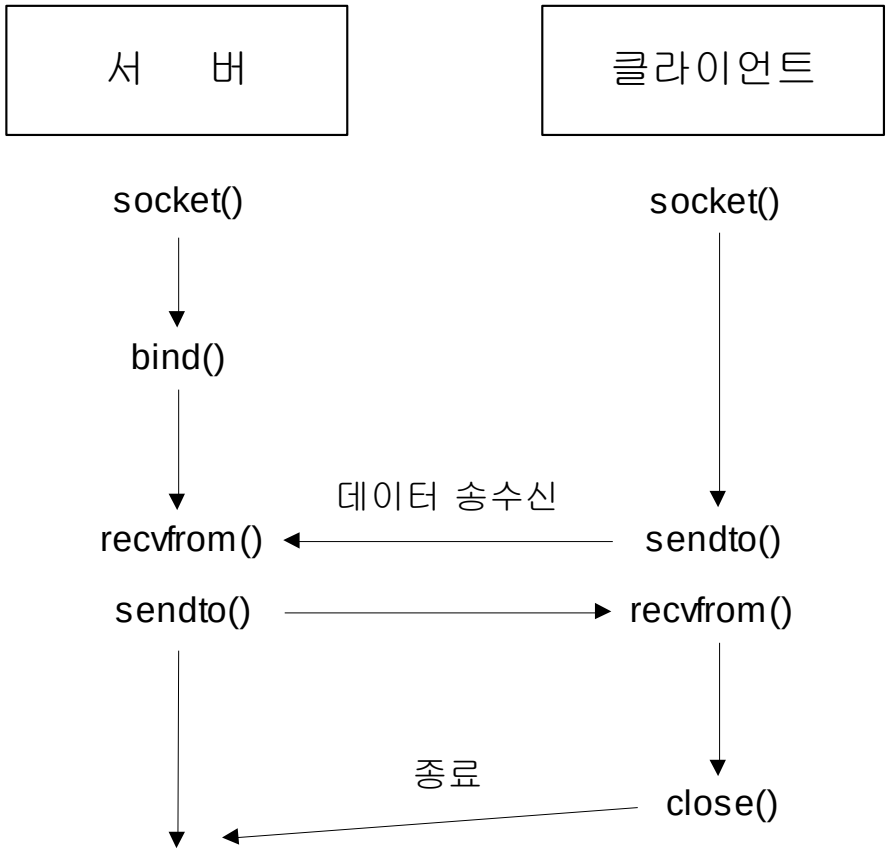
```
int recvfrom(int sock, int * buf, int len, unsigned flags,
                    struct sockaddr * addr, int * addrlen)
```

성공시 : 수신한 바이트 수
실패시 : -1 리턴

sock: 수신시 사용할 소켓 디스크립터를 인자로 넘김
buf : 수신할 데이터를 저장할 버퍼 포인터
len : 길이
flags : 0로 설정, 잘 사용되지 않음
addr : 데이터를 전송한 호스트의 주소로 채워짐
addrlen : 주소 구조체 변수길이

6.2 UDP기반 서버/클라이언트 구현

- 비연결 **UDP**기반의 함수 호출절차



6.2 UDP기반 서버/클라이언트 구현

- **비연결 UDP 방식에서 포트 및 주소할당**

  - 포트 번호는 TCP 소켓인 경우는 ***connect()*** 호출이 성공한 후에, UDP 소켓의 경우는 ***sendto()***가 성공한 후에 시스템 커널이 배정

  - UDP 기반 클라이언트 역시 bind() 함수 호출을 통해서 IP와 Port를 할당할 수 있다. 이 경우에는 sendto() 함수 호출시 IP와 Port가 재할당되지 않는다.

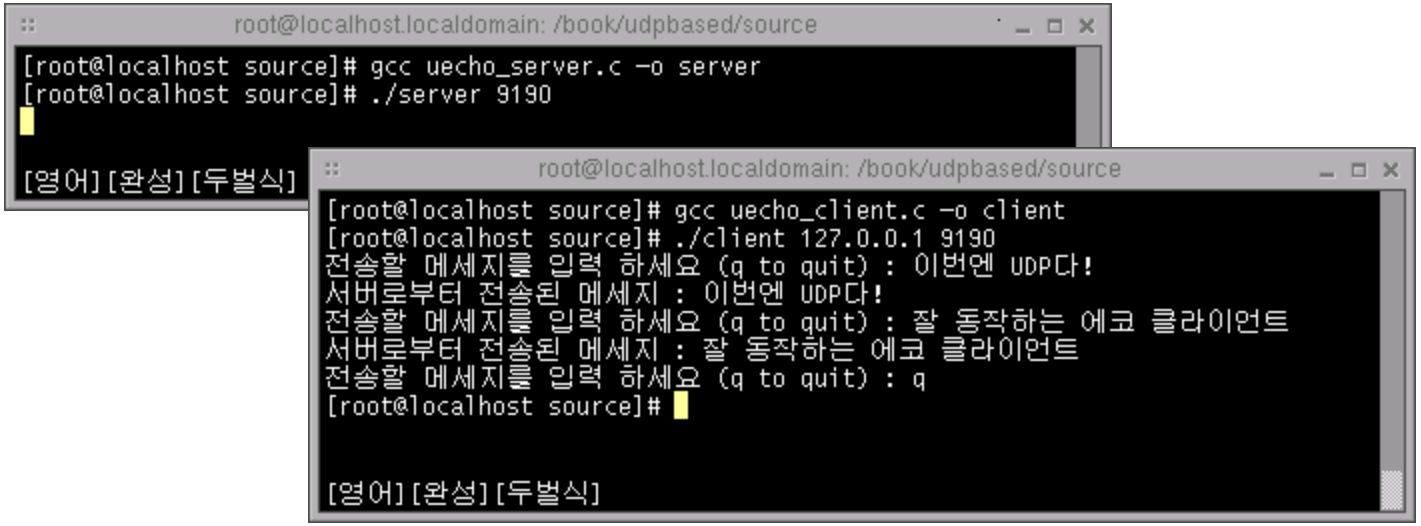
6.2 UDP기반 서버/클라이언트 구현

- **UDP기반의 Echo 서버/클라이언트**

1. 프로그램 예제

- uecho_server.c, uecho_client.c

2. 실행결과.



6.3 UDP 송수신 특성과 connect 함수 호출

- 데이터의 경계가 존재하는 **UDP**
  - 데이터 전송시 입출력 함수의 호출 횟수는 중요한 의미를 지님
  - **UDP** 소켓은 데이터를 송 수신하는데 필요한 함수 호출의 수를 서버, 클라이언트간에 정확히 일치시켜야 함

1. 프로그램 예제 및 실행결과
- bound_host1.c bound_host2.c

6.3 UDP 송수신 특성과 connect 함수 호출

- **UDP 소켓에서의 connect 함수의 의미**

1. **TCP 소켓에서의 connect 함수의 의미**
   - IP와 Port의 할당.
   - 연결 요청 진행(Three-way handshaking)

2. **UDP 소켓에서의 connect 함수의 의미**
   - 연결요청 과정은 생략,
   - IP와 Port의 할당하는 일만 수행

참 고 : connect 함수 호출을 하지 않으면 IP와 Port는 언제 할당 되는가?

- 포트 번호는 TCP 소켓인 경우는 ***connect()*** 호출이 성공한 후에,

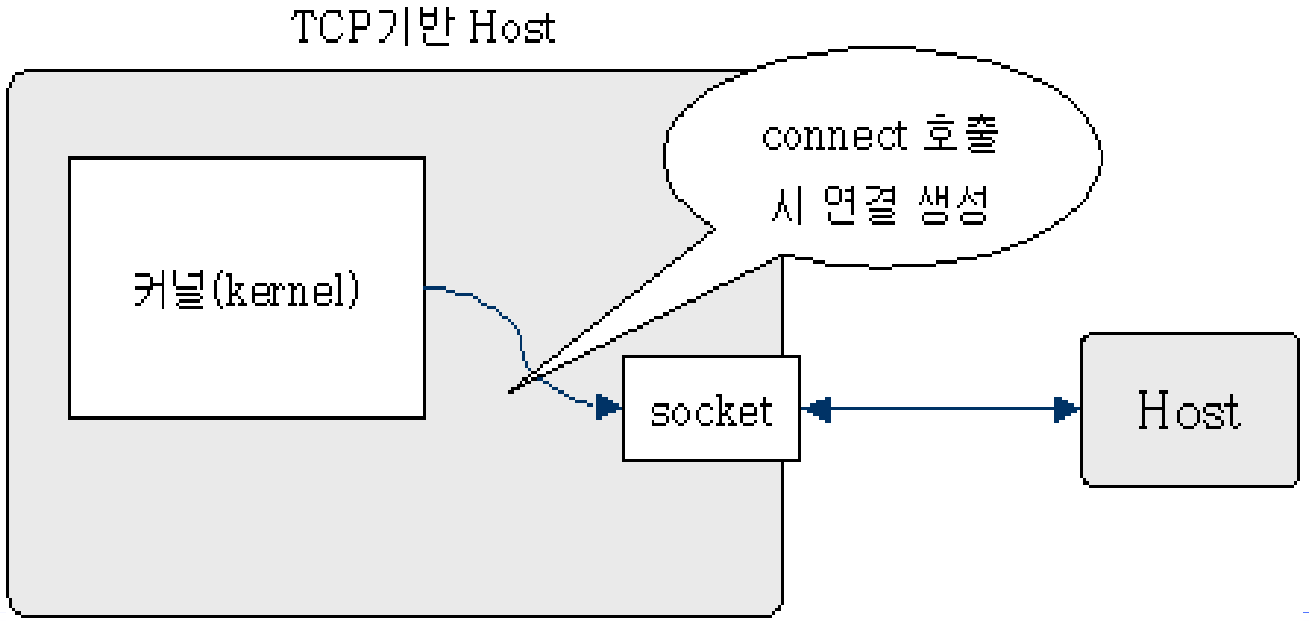
UDP 소켓의 경우는 ***sendto()***가 성공한 후에 시스템 커널이 배정

3. **TCP/UDP 소켓 공통적으로 지니는 connect 함수의 의미.**
   - 커널과 소켓의 연결 생성.

6.3 UDP 송수신 특성과 connect 함수 호출

- **connect 함수를 통해 얻어지는 잇점**

  - 커널에서 소켓의 연결을 유지
  - connect() 호출시 연결생성

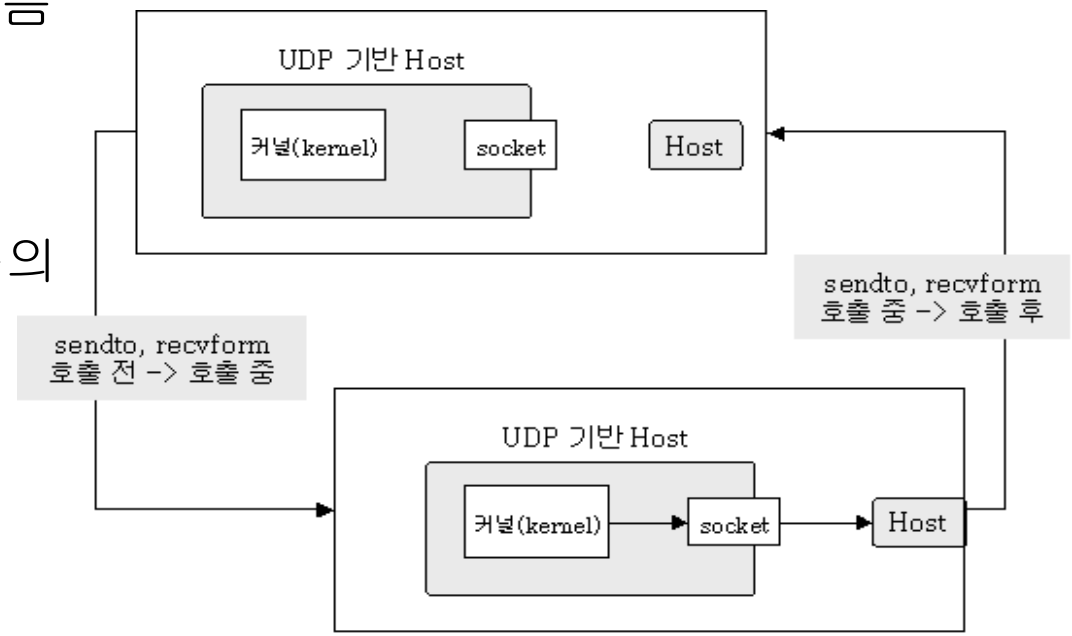


6.3 UDP 송수신 특성과 connect 함수 호출

- 일반적인 **UDP** 클라이언트

  - 함수 호출전에는 연결 없음
  - 함수 호출시 연결생성
  - 함수 종료 후 연결종료

  -이 절차가 전체 UDP 전송의 1/3을 차지



6.3 UDP 송수신 특성과 connect 함수 호출

- **UDP방식에서 connect() 호출이 주는 이점**
  - 데이터를 주고 받는 속도가 빨라진다.
  - **TCP** 소켓 기반의 데이터 입 출력 함수를 그대로 사용 할 수 있다.

    => read()/write() 사용가능
  - **connect()** 함수를 호출하여 **IP**와 **Port**를 할당

    => 연결을 계속 유지 가능함

6.3 UDP 송수신 특성과 connect 함수 호출

- 성능 향상된 **UDP**방식의 에코 클라이언트

1. 프로그램 예제

- 이전의 UDP echo 서버와 같이 동작
- connect() 함수 호출
- uecho_ con_client.c,

Q&A