# 13. IPC: Socket

Hyunchan, Park

http://oslab.jbnu.ac.kr

Division of Computer Science and Engineering

Jeonbuk National University

# 학습내용

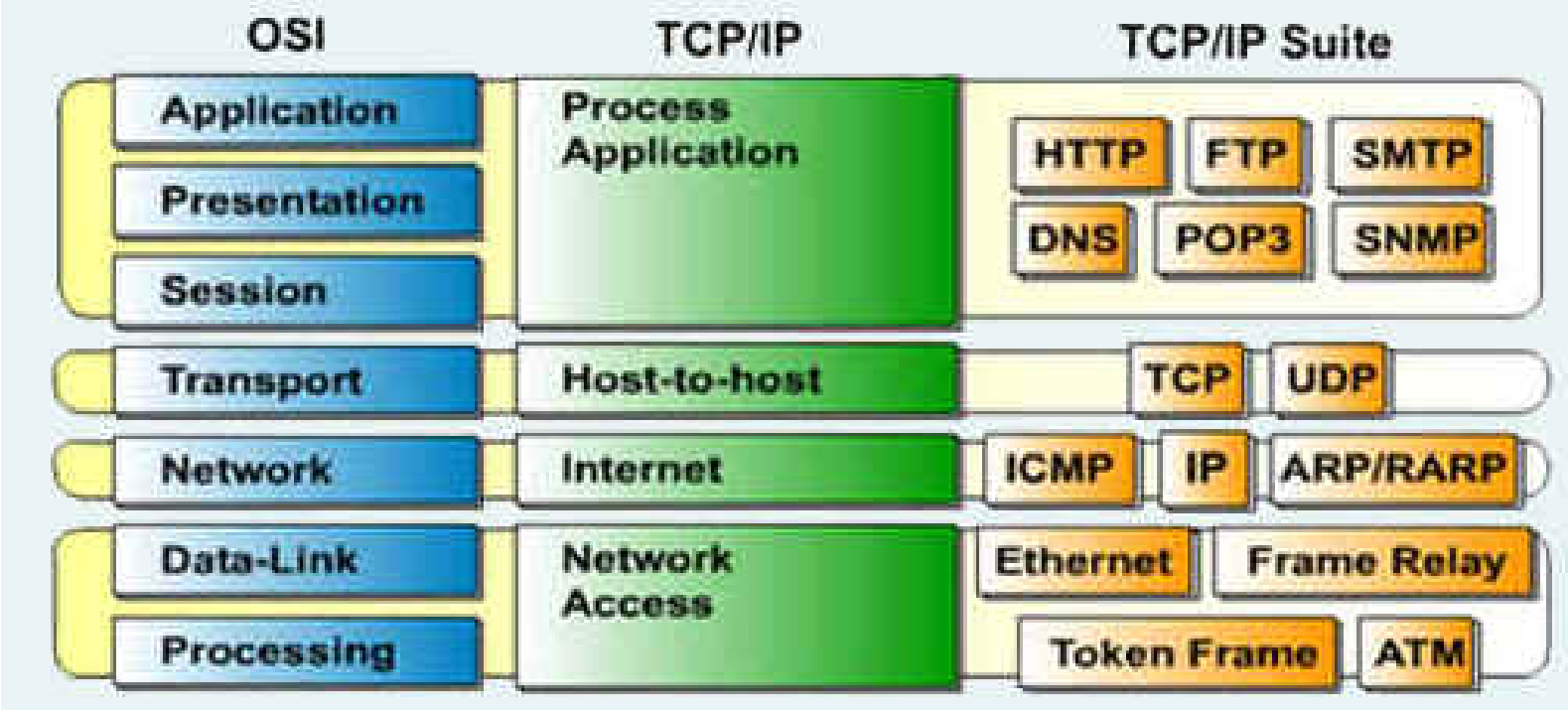
- TCP/IP Overview
- Socket

# TCP/IP Overview

# TCP/IP Overview

- TCP/IP
  - 인터넷 통신을 위한 표준 프로토콜
  - 4계층으로 구성
  - Packet 단위로 데이터를 주고 받음

| OSI 7 Layer | | TCP/IP 4 Layer | |
|---|---|---|---|
| L7 | 응용 계층 (Application Layer) | L4 | 응용 계층 (Application Layer) |
| L6 | 표현 계층 (Presentation Layer) | | |
| L5 | 세션 계층 (Session Layer) | | |
| L4 | 전송 계층 (Transport Layer) | L3 | 전송 계층 (Transport Layer) |
| L3 | 네트워크 계층 (Network Layer) | L2 | 인터넷 계층 (Internet Layer) |
| L2 | 데이터 링크 계층 (Data Link Layer) | L1 | 네트워크 엑세스 (Network Access Layer) |
| L1 | 물리 계층 (Physical Layer) | | |



# IP address and hostname

```
ubuntu@41983:~$ nslookup oslab.jbnu.ac.kr
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   oslab.jbnu.ac.kr
Address: 203.254.143.152
```

- IP주소와 호스트명
  - IP 주소: 네트워크에서 특정 시스템 혹은 노드 (node)를 식별하는 주소 체계
    - 점(.)으로 구분된 32비트 값 (192.168.0.1)
  - Host name: 로컬 네트워크에서 시스템에 부여된 이름
  - Domain name: 인터넷에서 IP 주소에 대해 Domain Name Service 에 의해 부여된 이름
    - nslookup 명령을 통해 domain name 이 가리키는 IP 주소를 알 수 있음
  - Special IP address 127.0.0.1 (loopback) : 자기 자신을 가리키는 특수한 IP 주소
- Public and Private IPs
  - Public IP: 전체 인터넷 망에서 각 시스템을 구분하는 주소
    - 전세계 어디서든 동일한 IP로 특정 시스템에 접속 가능 (예. 203.254.143.152)
  - Private IP: Private (or local) network 에서 각 시스템을 구분하는 주소
    - 예) 우리집 공유기 안에서만 노트북, 핸드폰, 스마트 TV를 구분 (192.168.0.xxx)
- 호스트 명과 IP주소 간의 변환
  - /etc/hosts 파일 또는 DNS 등
  - /etc/nsswitch.conf 파일에 주소변환을 누가 할 것인지 지정

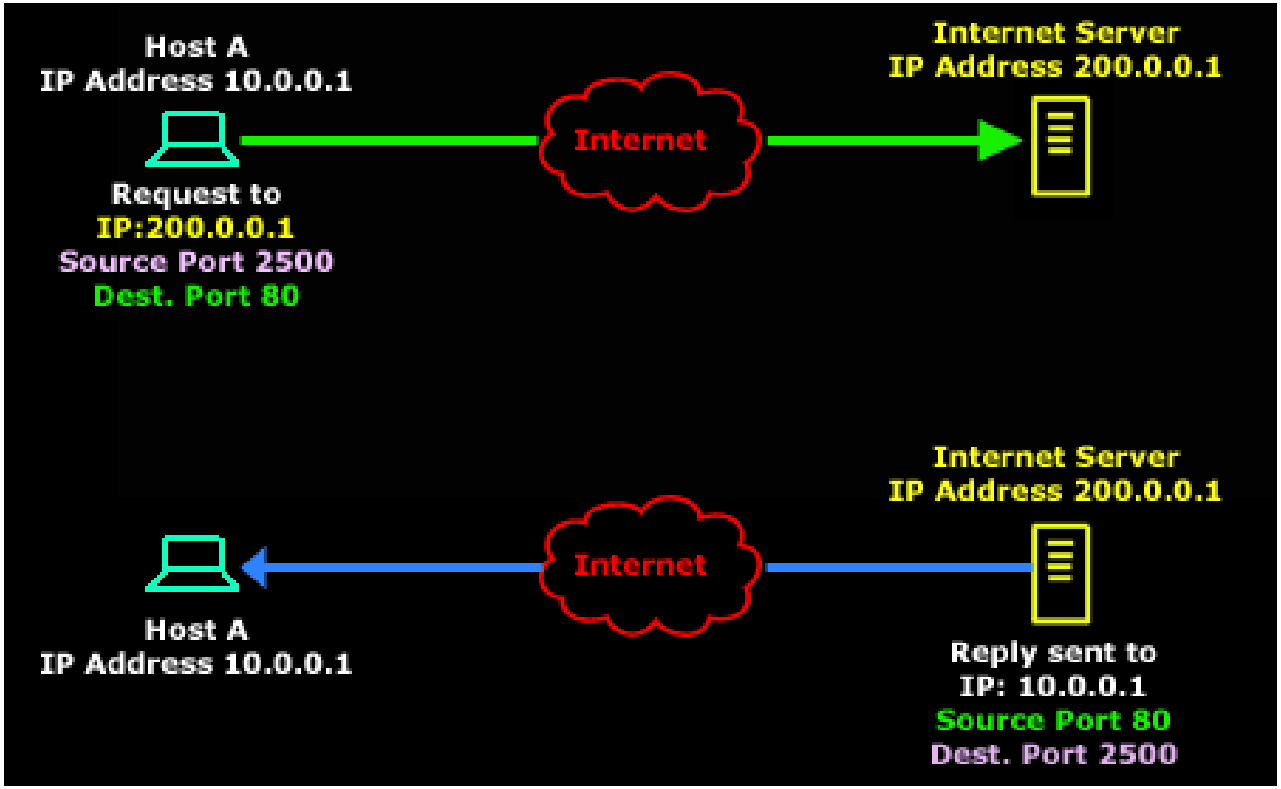
```
ubuntu@41983:~$ cat /etc/hosts
127.0.0.1 localhost
```

# Port

- Port: an endpoint of the network communication
  - 실제 통신을 수행하는 주체를 구분하는 번호
    - 주체: 호스트에서 동작하고 있는 서비스
    - 서비스: 한 프로세스가 여러 서비스(여러 포트를 통해)를 운영할 수도 있고, 하나의 서비스가 여러 프로세스를 이용해 동작할 수 있음
  - 2바이트 정수로 0~65535까지 사용 가능
  - <IP address>:<port number>형태로 전체 주소를 지정
    - 예) 203.254.143.152:80 -> 해당 호스트에서 동작하는 http 서비스
- 잘 알려진 포트
  - 잘 알려진 서비스들 (well-known services)의 포트 번호 (보통 1024 이하)
  - SSH(22), HTTP(80), FTP(21)
  - /etc/services : 잘 알려진 포트 번호들을 저장

# IP Network Communication

- IP 주소를 통해 네트워크 상에서 특정 호스트를 식별하고,
- Port 번호를 통해 호스트 내에서 특정 서비스를 식별하여,
- 원격 서비스들 간의 통신이 가능함



# TCP and UDP

- IP 상단의 전송 계층(L3)에서 통신을 위한 부가적인 기능을 제공하는 프로토콜들
  - IP 의 역할: 인터넷 상에서 특정 IP 주소를 가진 시스템을 찾아 패킷을 전해주는 것
  - TCP/UDP 의 역할: 단순히 패킷을 전달한다 라는 것을 넘어,
    패킷을 순서대로 전달한다거나, 패킷이 전달되었다는 것을 보장한다거나,
    패킷을 보내는 속도를 조절하여 성능을 극대화 하는 등의 부가 기능을 수행
  - 일반적으로 프로그램(L4)은 TCP 혹은 UDP 를 이용하여 실제 통신을 수행함
- UDP(User Datagram Protocol): 매우 단순한 패킷 전송 역할만 수행
- TCP(Transmission Control Protocol): 신뢰성있는 통신을 위한 다양한 기능 제공
  - 인터넷에서 TCP를 주로 사용하기 때문에, TCP/IP 라고 부름

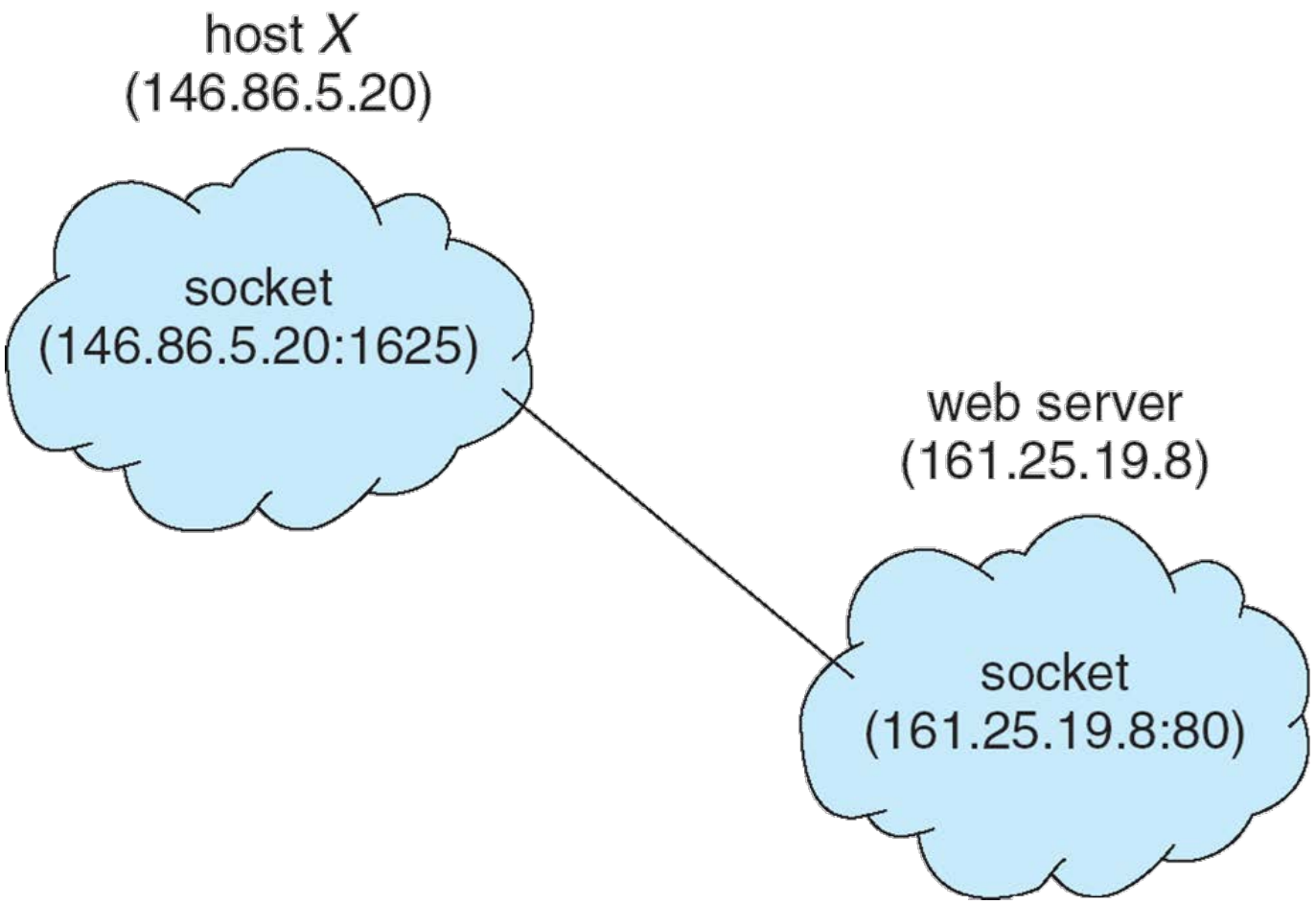
| TCP | UDP |
|---|---|
| 연결지향형(connection-oriented) | 비연결형(connectionless) |
| 신뢰성(reliability) 보장 | 신뢰성을 보장하지 않음 |
| 흐름 제어 기능(flow-control) 제공 | 흐름 제어 기능 없음 |
| 순서 보장(sequenced) | 순서를 보장하지 않음(no sequence) |

# Socket

# Socket

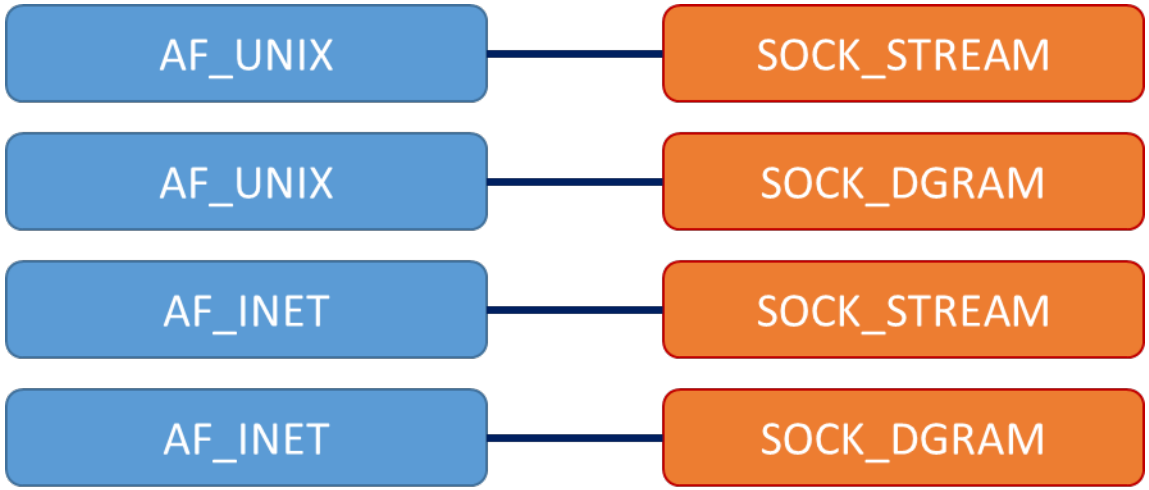
- Socket: an endpoint of the network communication (?!)
  - OS에서 제공하는 네트워크 통신을 위한 IPC abstraction
    - 소켓이 바로 IP:Port 를 이용해서 통신을 수행하는 주체
      - 예) 소켓=핸드폰, IP:port=핸드폰 번호
    - 즉, ip:port 는 결국 특정 소켓을 지정하는 이름이라 할 수 있음
- 프로그램은 고유의 포트 번호를 부여받은 소켓을 이용해서 네트워크 통신을 수행할 수 있음
  - 예) 친구랑 전화를 하고 싶으면?
    - 새 핸드폰을 산다 ≈ 새 소켓을 연다
    - 통신사 등록하여 개통한다 ≈ 소켓에 Port 번호를 부여함 (IP 주소는 이미 있음)
    - 전화를 건다 ≈ TCP 의 경우, 전화를 걸고, 받아서 연결이 수립되어야 통신 가능
    - 통화를 한다 ≈ 소켓을 이용해 패킷 단위로 데이터를 주고 받는다
    - (통신을 다 했으면? 소켓을 닫는다 ≈ 핸드폰을 버린다)

# Socket


host X
(146.86.5.20)
socket
(146.86.5.20:1625)
web server
(161.25.19.8)
socket
(161.25.19.8:80)

# 소켓: 종류 및 통신 방식

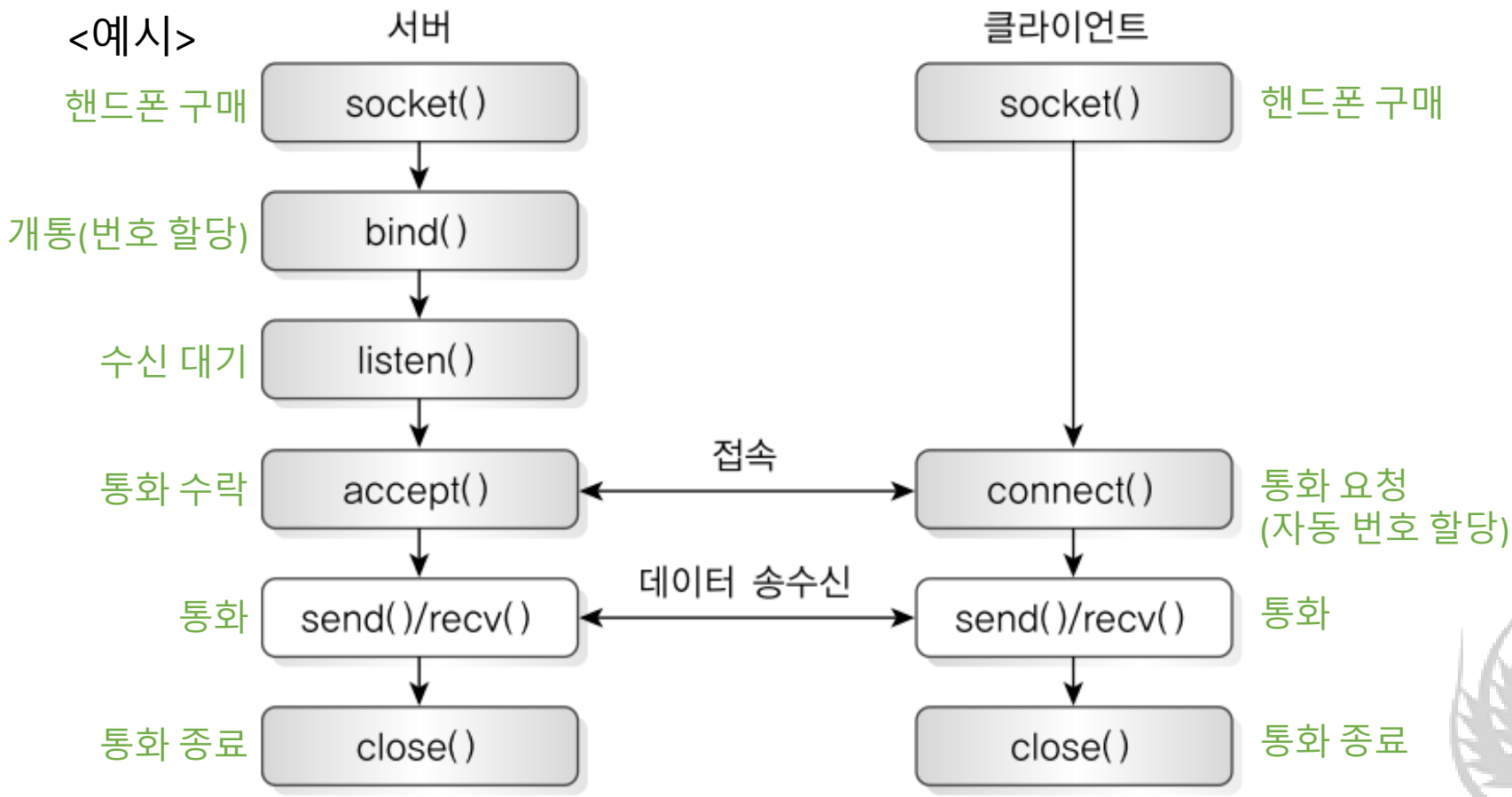
- 소켓의 종류
  - AF_UNIX : 유닉스 도메인 소켓 (시스템 내부 프로세스간 통신)
  - AF_INET : 인터넷 소켓 (네트워크를 이용한 통신)
- 소켓의 통신 방식
  - SOCK_STREAM : TCP 사용
  - SOCK_DGRAM : UDP 사용
- 즉, 아래와 같이 4가지 종류의 소켓을 사용할 수 있음
  - (AF_INET, SOCK_STREAM): 일반적인 TCP 를 이용한 인터넷 통신



# 소켓: 관련 시스템콜

- socket() : 소켓 파일 기술자 생성
- bind() : 소켓 파일 기술자를 지정된 IP 주소/포트번호와 결합(bind)
- listen() : 클라이언트의 접속 요청 대기
- connect() : 클라이언트가 서버에 접속 요청
- accept() : 클라이언트의 접속 허용
- recv() : 데이터 수신(SOCK_STREAM)
- send() : 데이터 송신(SOCK_STREAM)
- recvfrom() : 데이터 수신(SOCK_DGRAM)
- sendto() : 데이터 송신(SOCK_DGRAM)
- close() : 소켓 파일기술자 종료

# 소켓: TCP 통신을 위한 수행 순서



# 1. 소켓 생성: socket(2)

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- domain : 소켓 종류 (AF_UNIX, AF_INET)
- type : 통신 방식 (TCP, UDP)
- protocol : 소켓에 이용할 프로토콜
  - 0: 지정한 통신 방식의 기본 프로토콜
- Return value: 새롭게 생성된 소켓의 file descriptor
- 소켓 또한 파일 형태로 관리가 됨
- 따라서 소켓을 모두 이용한 다음에는 close()를 이용해서 닫기

```
int sd;
sd = socket(AF_INET, SOCK_STREAM, 0);

close(sd);
```

# 2. 소켓 이름 지정: bind(3)

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

- sockfd: socket file descriptor
- addr : 소켓의 이름을 표현하는 구조체 sockaddr 의 주소
- addrlen: struct sockaddr 의 크기

```
int sd = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in sin;
memset((char *)&sin, '\0', sizeof(sin));

sin.sin_family = AF_INET;
sin.sin_port = htons(9000);
sin.sin_addr.s_addr = inet_addr("192.168.100.1");

bind(sd, (struct sockaddr *)&sin, sizeof(struct sockaddr));
```

# * 네트워크 주소를 표현하는 구조체

- 인터넷 소켓의 주소 구조체

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

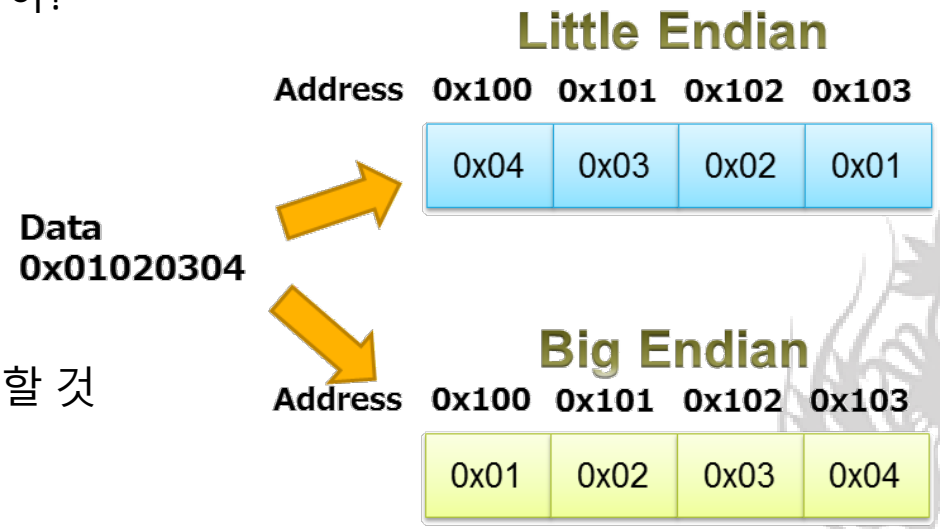
- 유닉스 도메인 소켓의 주소 구조체

```
struct sockaddr_un {
    sa_family_t
sun_famyly;
    char
sun_path[108]
};
```

- **sun_family : AF_UNIX**
- **sun_path : 통신에 사용할 파일의 경로명**

# * 바이트 저장 순서 지정: endian

- Endian
  - 정수를 저장하는 방식 : Big and Little endian
  - Little endian: 메모리의 높은 주소에 정수의 첫 바이트를 위치 -> 인텔, ARM
  - Big endian: 메모리의 낮은 주소에 정수의 첫 바이트를 위치 -> 모토로라, 썬
- 인텔 머신과 썬 머신이 서로 통신을 하면???
  - 인텔: "야 내가 0x0102 (258) 이라고 했잖아!"
  - 썬: "너 0x0201 (513) 라고 했거든!!"
- TCP/IP 표준: Big Endian
  - Host byte order(HBO)
  - Network byte order(NBO)
  - HBO와 NBO가 항시 다를 수 있음을 기억할 것



# * 바이트 저장 순서 지정: 관련 서비스

```
#include <arpa/inet.h>

uint32_t htonl(unit32_t hostlong);
uint32_t ntohl(unit32_t netlong);
uint16_t htons(unit16_t hostshort);
uint16_t ntohs(unit16_t netshort);
```

- htonl :32비트 HBO를 32비트 NBO로 변환
- ntohl : 32비트 NBO를 32비트 HBO로 변환
- htons : 16비트 HBO를 16비트 NBO로 변환
- ntohs : 16비트 NBO를 16비트 HBO로 변환

- Endian 을 변환해주어야 하는 정보
  - Type 이 지정되어 저장되는 정보
    - IP주소, 포트 번호 (int port = 8080;)
  - 실제 데이터를 주고 받을 때는 문자열 배열과 같은 형태의 바이트 집합을 주고받기 때문에 변환이 불필요함 (unsigned char buf[80];)

# * 네트워크 주소 변환 서비스

- IP주소의 형태
  - 192.168.10.1과 같이 점(.)으로 구분된 형태
  - 시스템 내부 저장 방법 : 이진값으로 바꿔서 저장
  - 외부에서는 일반적으로 문자열로 표기

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

in_addr_t inet_addr(const char *cp);
char *inet_ntoa(const struct in_addr in);
```

- 문자열 형태의 IP주소를 NBO 32비트 값으로 변환 : inet_addr(3)
- 구조체 형태의 NBO IP주소를 문자열 형태로 변환: inet_ntoa(3)

# 3. 연결 대기 및 수락: listen(3) and accept(3)

```
#include <sys/types.h>
#include <sys/socket.h>

int listen(int sockfd, int backlog);
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- 클라이언트 연결 기다리기: listen(3)
  - backlog : 동시 접속을 허용할 최대 클라이언트 수
- 연결 요청 수락하기: accept(3)
  - addr, addrlen: 접속을 요청한 클라이언트의 IP 정보를 저장할 메모리 주소들
    - Addrlen 은 반드시 구조체 크기 값으로 설정을 해주어야 함
  - Return value: 수락된 연결에 대한 새로운 소켓을 반환
    - 다수 클라이언트와 동시에 통신하기 위해,
    - Sockfd 소켓은 또다른 연결 요청을 대기하기 위해 사용하고,
    - 새로운 소켓을 이용해 클라이언트와 통신을 수행함
- UDP의 경우, 비연결성 프로토콜이기 때문에 “연결” 동작이 불필요함
  - 따라서 listen(), accept(), connect() 를 사용하지 않음

# 4. 연결 요청: connect(3)

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr, int addrlen);
```

- addr : 접속하려는 서버의 IP정보
- addrlen: addr 구조체의 크기

```
int sd = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in sin;
memset((char *)&sin, '\0', sizeof(sin));

sin.sin_family = AF_INET;
sin.sin_port = htons(9000);
sin.sin_addr.s_addr = inet_addr("192.168.100.1");

connect(sd, (struct sockaddr *)&sin, sizeof(struct sockaddr));
```

# 5. TCP 데이터 송수신: send(3) and recv(3)

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t send(int sockfd, const void *buf, size_t len, int flags);

ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

- buf, len: 데이터가 기록된/기록될 공간의 주소와 크기
- flags: IO 블록킹, 에러 처리 등의 옵션을 쓸 수 있음. 보통 0.
- Return value: 전송한/수신한 데이터의 크기

```
char *msg = "Send Test\n";
int len = strlen(msg) + 1;          //문자열 종료 문자 '\0' 를 포함시키기 위함
if (send(sd, msg, len, 0) == -1) {
    perror("send");
    exit(1);
}
```

```
char buf[80];
int len=sizeof(buf), rlen=0;   //buf[]의 크기를 recv()에 전달해 overflow 방지
if ((rlen = recv(sd, buf, len, 0)) == -1) {
    perror("recv");
    exit(1);
}
```

# 6. UDP 데이터 송수신: sendto(3) and recvfrom(3)

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
                   const struct sockaddr *dest_addr, socklen_t addrlen);

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                   struct sockaddr *src_addr, socklen_t *addrlen);
```

- TCP의 send(), recv()와 다른 점
  - UDP는 비연결성 프로토콜이기 때문에,
  - 매 호출 시마다 보낼 주소, 받을 주소를 명시하여야 함
  - 1회성으로 정보를 다수 서버에 전송할 경우, 효율적임

# [예제 1] 간단한 메시지 전송: Server (1/2)

hw11 > C n1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define BUFFSIZE 4096
#define SERVERPORT 7799

int main(void) {
    int s_sock, c_sock;
    struct sockaddr_in server_addr, client_addr;
    socklen_t c_addr_size;
    char buf[BUFFSIZE] = {0};
    char hello[] = "Hello~I am Server!\n";

    s_sock = socket(AF_INET, SOCK_STREAM, 0);

    bzero(&server_addr, sizeof(server_addr));

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVERPORT);
    //server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_addr.sin_addr.s_addr = inet_addr("10.0.0.249");

    if (bind(s_sock, (struct sockaddr *) &server_addr, sizeof(server_addr)) == -1) {
        perror("Can't bind a socket");
        exit(1);
    }

    listen(s_sock,1);
```

# [예제 1] 간단한 메시지 전송: Server (2/2)

```
33      listen(s_sock,1);
34
35      c_addr_size = sizeof(struct sockaddr);
36      c_sock = accept(s_sock, (struct sockaddr *) &client_addr, &c_addr_size);
37      if (c_sock == -1) {
38          perror("Can't accept a connection");
39          exit(1);
40      }
41
42      printf("** Check: s_sock=%d c_sock=%d\n", s_sock, c_sock);
43      printf("** Check: client IP addr=%s port=%d\n", inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
44
45      //1. say hello to client
46      if (send(c_sock, hello, sizeof(hello)+1, 0) == -1) {
47          perror("Can't send message");
48          exit(1);
49      }
50
51      printf("I said Hello to Client!\n");
52
53      //2. recv msg from client
54      if (recv(c_sock, buf, BUFFSIZE, 0) == -1) {
55          perror("Can't receive message");
56          exit(1);
57      }
58
59      printf("Client says: %s\n", buf);
60
61      close(c_sock);
62      close(s_sock);
63
64      return 0;
65  }
```

# [예제 1] 간단한 메시지 전송: Client (1/2)

hw11 > C n2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define BUFFSIZE 4096
#define SERVERPORT 7799

int main(void) {
    int c_sock;
    struct sockaddr_in server_addr, client_addr;
    socklen_t c_addr_size;
    char buf[BUFFSIZE] = {0};
    char hello[] = "Hi~I am Client!!\n";

    c_sock = socket(AF_INET, SOCK_STREAM, 0);

    bzero(&server_addr, sizeof(server_addr));

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVERPORT);
    //server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_addr.sin_addr.s_addr = inet_addr("10.0.0.249");

    if (connect(c_sock, (struct sockaddr *) &server_addr, sizeof(server_addr)) == -1) {
        perror("Can't connect to a Server");
        exit(1);
    }

    printf("Check: c_sock=%d\n", c_sock);
```

# [예제 1] 간단한 메시지 전송: Client (2/2)

```
33      printf("Check: c_sock=%d\n", c_sock);
34
35      //1. recv msg from server (maybe it's "hello")
36      if (recv(c_sock, buf, BUFFSIZE, 0) == -1) {
37          perror("Can't receive message");
38          exit(1);
39      }
40
41      printf("Server says: %s\n", buf);
42
43      //2. say hi to server
44      if (send(c_sock, hello, sizeof(hello)+1, 0) == -1) {
45          perror("Can't send message");
46          exit(1);
47      }
48
49      printf("I said Hi to Server!!\n");
50
51      close(c_sock);
52
53      return 0;
54  }
```

# [예제 1] 결과

```
ubuntu@41983:~/hw11$ gcc -o n1 n1.c
ubuntu@41983:~/hw11$ ./n1
** Check: s_sock=3 c_sock=4
** Check: c_addr_size=16
** Check: client IP addr=10.0.0.249 port=36092
I said Hello to Client!
Client says: Hi~I am Client!!
```

```
ubuntu@41983:~/hw11$ gcc -o n2 n2.c
ubuntu@41983:~/hw11$ ./n2
Check: c_sock=3
Server says: Hello~I am Server!

I said Hi to Server!!
ubuntu@41983:~/hw11$
```

# 개인 과제 11: file transfer server and client

- 프로그램 2개 작성: server and client
  - hw11s.c: 파일 하나를 요청받고, 해당 파일을 전송해주는 프로그램
    - 명령행 인자: 없음
  - hw11c.c: 파일 하나를 요청하고, 서버로부터 해당 파일을 수신하는 프로그램
    - 명령행 인자: 접속할 서버의 IP주소, 포트 번호, 요청할 파일명 (경로는 생략하고 이름만)
- 주의 사항
  - /home/ubuntu/hw11 에서 위 파일명으로 새로운 파일들 생성하여 작업할 것
  - Server는 동작 중인 호스트 시스템의 IP 주소를 자동으로 받아와서 bind() 에 사용할 것
    - 다양한 방식이 있으므로 보고서에 참고한 자료, 사용한 방식과 설명을 명시할 것
  - 파일의 크기에 대한 가정은 없음. 매우 큰 파일의 전송에도 문제없이 동작할 것
  - 파일의 이름과 크기 외의 다른 정보(권한, 접근 시간 등)은 전송하지 않아도 됨
  - 결과에는 diff 명령을 이용하여 수신한 파일이 원본과 동일한지 검사한 결과 캡처를 포함할 것
- 제출 기한
  - 12/7 (월) 23:59 (지각 감점: 5%p / 12H, 1주 이후 제출 불가)
  - 동작 설명과 결과가 포함된 간단한 보고서(PDF!!!)와 소스 파일들을 압축하여 LMS 제출