Chapter 06

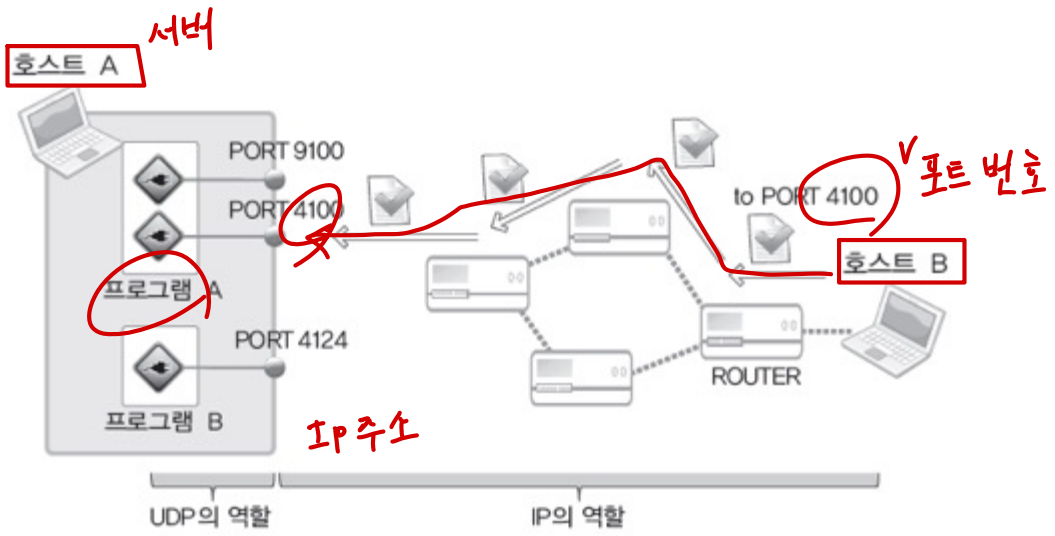
UDP 기반 서버/클라이언트

UDP 소켓의 특성과 동작원리

- UDP 소켓과 TCP 소켓의 데이터 송수신 비교
  - UDP 소켓은 SEQ, ACK와 같은 메시지를 전달하지 않음: Flow control이 없음
  - 연결 설정과 연결 해제 과정도 없음
    - 데이터 분실 및 손실의 위험이 있음
  - 데이터 전송시 확인 과정이 존재하지 않기 때문에 데이터 전송이 빠름
  - 안전성보다 성능이 우선시 될 때에 UDP 사용
  - 송수신하는 데이터의 양은 작으면서 빈번한 전송이 필요한 경우 UDP가 훨씬 효율적임

기준이 뭘까?

UDP 역할
- 호스트로 전달된 패킷을 PORT 정보를 참조하여 최종 목적지인 UDP 소켓에 전달

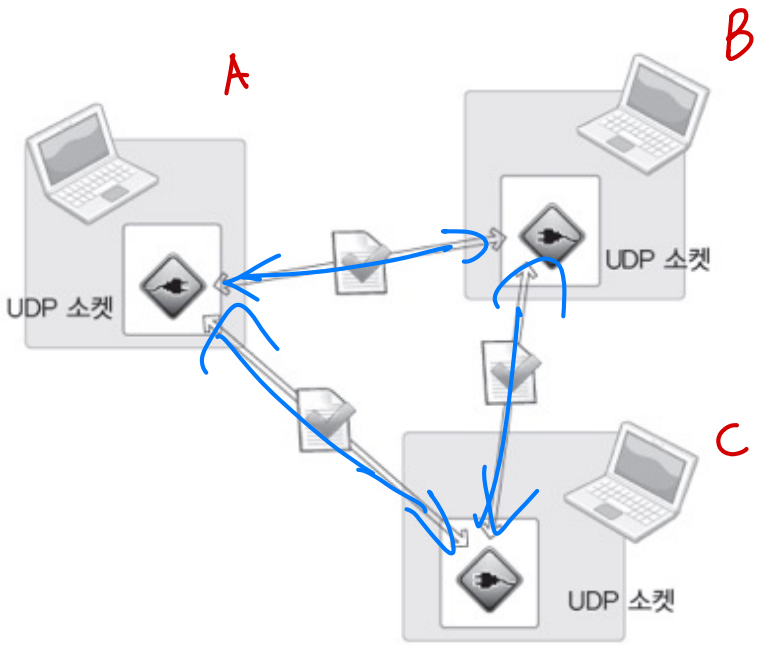


▶ 그림 06-1 : 패킷 전송에 있어서의 UDP와 IP의 역할

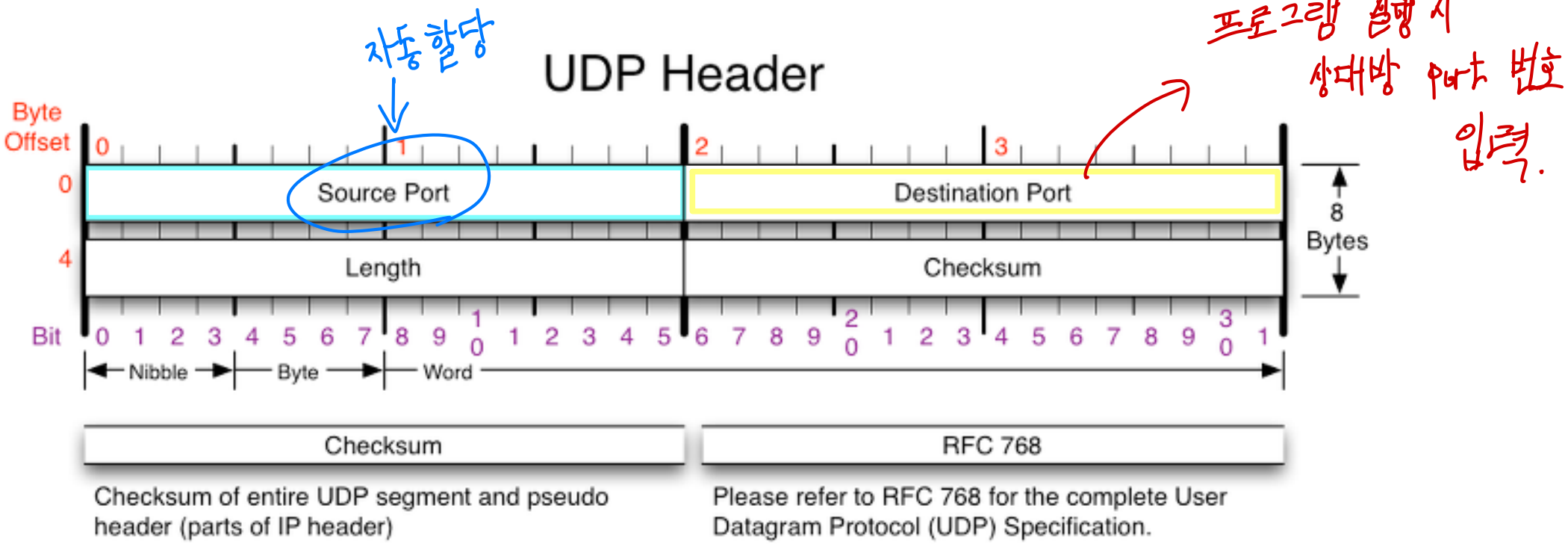
UDP 소켓의 데이터 송수신

- UDP의 데이터 송수신
  - UDP는 연결의 개념이 존재하지 않음
    - TCP는 1대 1의 연결을 필요
  - 서버 소켓과 클라이언트 소켓의 구분이 없음
    - UDP 소켓 생성과 데이터 송수신 과정만 존재함
  - 하나의 소켓으로 둘 이상의 호스트와 데이터 송수신이 가능함
    - 그룹 통신: 14장 멀티캐스트에 사용됨

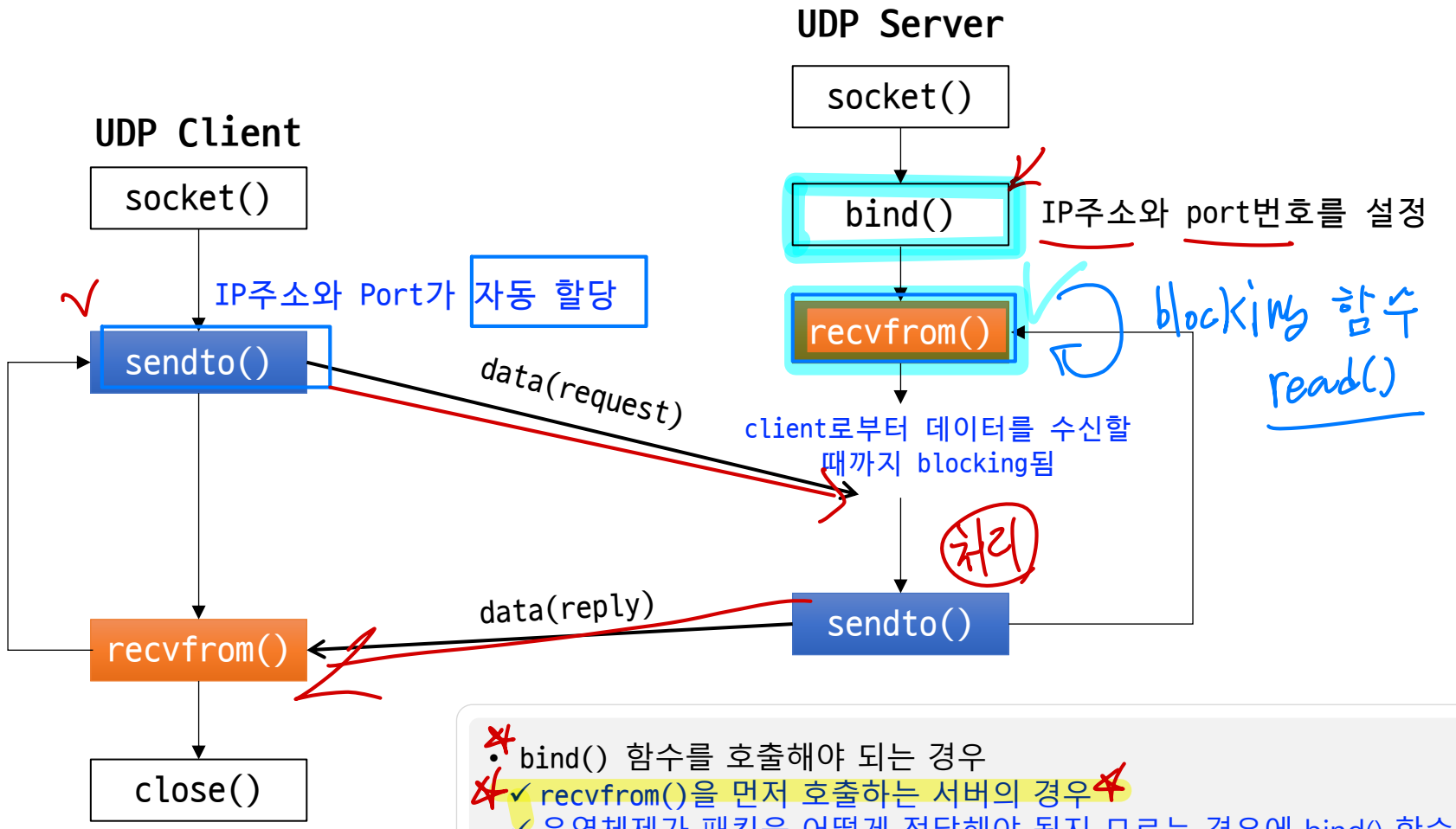
= broadcasting



UDP Header


UDP Header
자동할당
프로그램 실행 시
상대방 port 번호
입력.
Byte Offset
0
4
Bit
Source Port
Destination Port
Length
Checksum
8 Bytes
Nibble
Byte
Word
Checksum
RFC 768
Checksum of entire UDP segment and pseudo header (parts of IP header)
Please refer to RFC 768 for the complete User Datagram Protocol (UDP) Specification.
Copyright 2008 - Matt Baxter - mjb@fatpipe.org - www.fatpipe.org/~mjb/Drawings/

UDP 통신 흐름



- bind() 함수를 호출해야 되는 경우
  - ✓ recvfrom()을 먼저 호출하는 서버의 경우
  - ✓ 운영체제가 패킷을 어떻게 전달해야 될지 모르는 경우에 bind() 함수를 호출

- bind() 함수를 호출하지 않아도 되는 경우는 sendto()를 먼저 호출하는 경우임
  - ✓ sendto() 함수를 호출할 때 인자로 주어지는 상대방 주소 정보를 참조하여 운영체제가 자동으로 bind를 해줌

UDP 기반의 데이터 입출력 함수 #1

- sendto() 함수
  - UDP 소켓은 연결 개념이 없음
    - 데이터를 전송할 때마다 목적지에 대한 정보를 전달해야 됨

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

ssize_t sendto(int sock, void *buf, size_t nbytes, int flags,
               struct sockaddr *to, socklen_t addrlen);
   -> 성공 시 전송된 바이트 수, 실패 시 -1 반환
```

- sock: 데이터 전송에 사용될 UDP 소켓
- buf: 전송할 데이터를 저장하고 있는 버퍼의 주소값
- nbytes: 전송할 데이터의 크기
- flags: 옵션 지정에 사용, 지정할 옵션이 없으면 0 입력
- to: 목적지 주소 정보를 담고 있는 구조체 변수의 주소값
- addrlen: to로 전달될 구조체(sockaddr)의 크기

UDP 기반의 데이터 입출력 함수 #2

- recvfrom() 함수
  - 데이터의 전송지가 둘 이상이 될 수 있음
    - 데이터 수신 후 전송지가 어디인지 확인이 필요함

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

ssize_t recvfrom(int sock, void *buf, size_t nbytes, int flags,
                 struct sockaddr *from, socklen_t *addrlen);
   -> 성공 시 수신한 바이트 수, 실패 시 -1 반환
```

- sock: 데이터 수신에 사용될 UDP 소켓의 파일 디스크립터
- buf: 데이터 수신에 사용될 버퍼의 주소값
- nbytes: 수신할 최대 바이트 수, buf의 크기를 넘을 수 없음
- flags: 옵션 지정에 사용, 옵션이 없으면 0 전달
- from: 발신지 주소 정보를 저장할 sockaddr 구조체 주소값
- addrlen: from으로 전달될 주소값의 구조체(sockaddr) 변수 크기

UDP 기반의 에코 서버와 클라이언트

UDP 에코 서버

```
if(bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1)
    error_handling("bind() error");

while(1)
{
    clnt_adr_sz=sizeof(clnt_adr);
    str_len = recvfrom(serv_sock, message, BUF_SIZE, 0, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);

    sendto(serv_sock, message, str_len, 0, (struct sockaddr*)&clnt_adr, clnt_adr_sz);
}
```

UDP 에코 서버 코드에서는 수신한 데이터의 전송지 정보를 참조하여 데이터를 전송(echo) 함

UDP 에코 클라이언트

```
while(1)
{
    fputs("Insert message(q to quit): ", stdout);
    fgets(message, sizeof(message), stdin);
    if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
        break;
    sendto(sock, message, strlen(message), 0, (struct sockaddr*)&serv_adr, sizeof(serv_adr));

    adr_sz = sizeof(from_adr);
    str_len = recvfrom(sock, message, BUF_SIZE, 0, (struct sockaddr*)&from_adr, &adr_sz);

    message[str_len]=0;
    printf("Message from server: %s", message);
}
```

- UDP는 데이터의 경계가 존재하기 때문에 한번의 recvfrom() 함수호출을 통해서 하나의 메시지를 완전히 읽어 들임
- sendto() 함수호출 시 IP와 PORT번호가 자동으로 할당되기 때문에 일반적으로 UDP의 클라이언트 프로그램에서는 자신의 주소정보를 할당하는 별도의 과정이 없음

uecho_server.c #1

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

#define BUF_SIZE 30
void error_handling(char *message);

int main(int argc, char *argv[])
{
    int serv_sock;
    char message[BUF_SIZE];
    int str_len;
    socklen_t clnt_adr_sz;
    struct sockaddr_in serv_adr, clnt_adr;

    if(argc!=2){
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }

    serv_sock=socket(PF_INET, SOCK_DGRAM, 0);
    if(serv_sock==-1)
        error_handling("UDP socket creation error");
```

UDP 소켓 생성 (SOCK_DGRAM)

uecho_server.c #2

```
    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_adr.sin_port=htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1)
        error_handling("bind() error");

    while(1)
    {
        clnt_adr_sz=sizeof(clnt_adr);
        str_len=recvfrom(serv_sock, message, BUF_SIZE, 0,
                         (struct sockaddr*)&clnt_adr, &clnt_adr_sz);

        sendto(serv_sock, message, str_len, 0,
                      (struct sockaddr*)&clnt_adr, clnt_adr_sz);
    }
    close(serv_sock);
    return 0;
}

void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

recvfrom()

⬇

sendto()

recvfrom()을 먼저 호출하는 곳에서 bind()

clnt_adr에 클라이언트의 주소 정보가 저장됨

str_len 크기만큼 클라이언트에게 전송

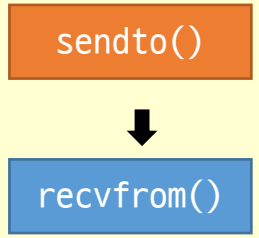
uecho_client.c #1

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

#define BUF_SIZE 30
void error_handling(char *message);

int main(int argc, char *argv[])
{
    int sock;
    char message[BUF_SIZE];
    int str_len;
    socklen_t adr_sz;
    struct sockaddr_in serv_adr, from_adr;
    if(argc!=3){
        printf("Usage : %s <IP> <port>\n", argv[0]);
        exit(1);
    }
    sock=socket(PF_INET, SOCK_DGRAM, 0);
    if(sock==-1)
        error_handling("socket() error");

    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=inet_addr(argv[1]);
    serv_adr.sin_port=htons(atoi(argv[2]));
```

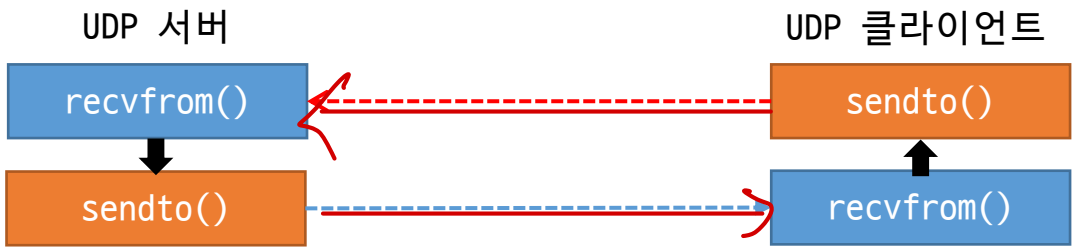
UDP 소켓 생성

uecho_client.c #2

```
    while(1)
    {
        fputs("Insert message(q to quit): ", stdout);
        fgets(message, sizeof(message), stdin);
        if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
            break;

        sendto(sock, message, strlen(message), 0,
                    (struct sockaddr*)&serv_adr, sizeof(serv_adr));

        adr_sz=sizeof(from_adr);
        str_len=recvfrom(sock, message, BUF_SIZE, 0,
                            (struct sockaddr*)&from_adr, &adr_sz);

        message[str_len]=0; // 수신된 문자열의 마지막에 NULL문자 추가
        printf("Message from server: %s", message);
    }
    close(sock);
    return 0;
}

void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```



실행 결과



- 서버 실행

```
$ gcc uecho_server.c -o userver
$ ./userver 9190
```

  - recvfrom()을 수행하는 프로그램이 먼저 수행되어야 함

- 클라이언트 실행

```
$ gcc uecho_client.c -o uclient
$ ./uclient 127.0.0.1 9190
Insert message(q to quit): Hi UDP Server?
Message from server: Hi UDP Server?
Insert message(q to quit): Nice to meet you!
Message from server: Nice to meet you!
Insert message(q to quit): q
```

netstat 명령어

- 시스템의 네트워크 연결 목록(tcp, udp)을 보여주는 유틸리티
- 옵션
  - -a: 현재 다른 PC와 연결되어 있거나 대기(listening)중인 모든 포트번호를 확인
  - -t: TCP Protocol
  - -u: UDP Protocol
  - -s: IP, ICMP, UDP 프로토콜별 상태를 보여줌
  - -nap: 열려 있는 모든 포트를 보여줌

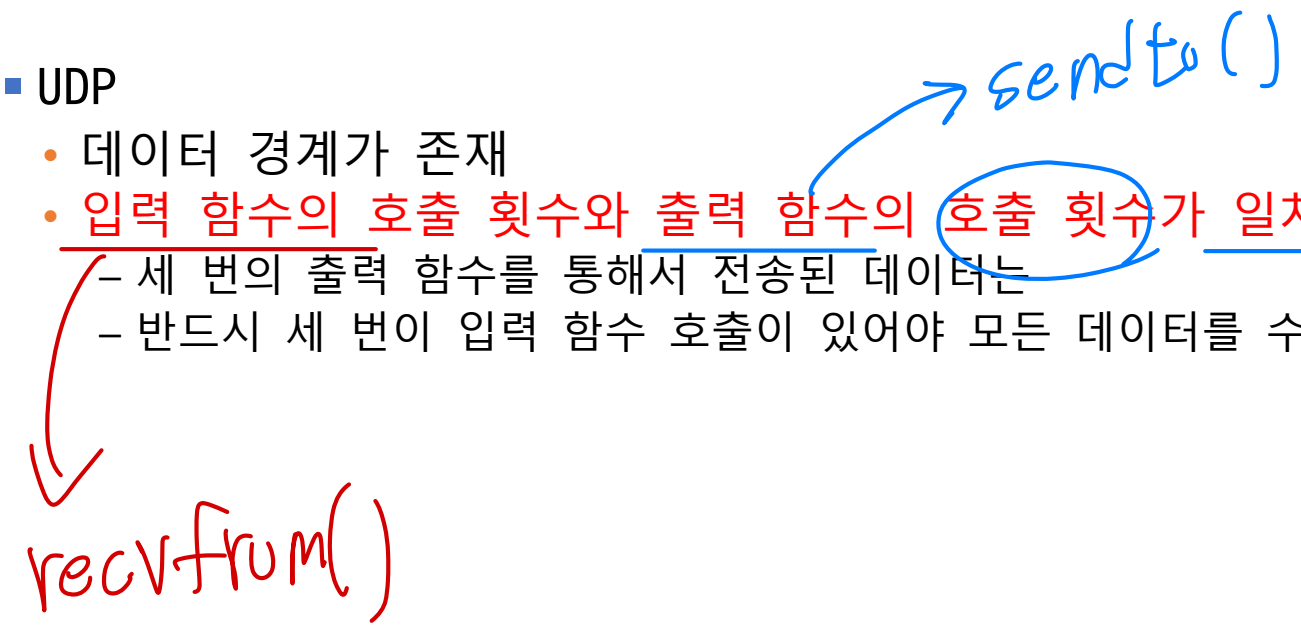
```
pi@raspberrypi:~ $ netstat -u -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 *:9190                  *:*
udp        0      0 *:35327                 *:*
udp        0      0 *:bootpc                *:*
udp        0      0 155.230.35.251:ntp      *:*
udp        0      0 localhost:ntp           *:*
udp        0      0 *:ntp                   *:*
udp        0      0 *:43209                 *:*
udp        0      0 *:mdns                  *:*
udp6       0      0 [::]:42599              [::]:*
udp6       0      0 fe80::ba27:ebff:fe4:ntp [::]:*
udp6       0      0 fe80::ba27:ebff:fee:ntp [::]:*
```

uecho_server를 실행한 상태에서 검사한 결과

데이터의 경계가 존재하는 UDP

- TCP
  - 송수신하는 데이터에는 경계가 존재하지 않음
  - 데이터 송수신 과정에서 입출력 함수의 호출 횟수는 큰 의미를 가지지 않음

- UDP
  - 데이터 경계가 존재
  - 입력 함수의 호출 횟수와 출력 함수의 호출 횟수가 일치해야 됨
    - 세 번의 출력 함수를 통해서 전송된 데이터는
    - 반드시 세 번이 입력 함수 호출이 있어야 모든 데이터를 수신할 수 있음



데이터의 경계가 존재하는 UDP 소켓

- bound_host1.c 데이터 수신 부분

```
if(bind(sock, (struct sockaddr*)&my_adr, sizeof(my_adr))==-1)
    error_handling("bind() error");

for(i=0; i<3; i++)
{
    sleep(5); // delay 5 sec.
    adr_sz=sizeof(your_adr);
    str_len=recvfrom(sock, message, BUF_SIZE, 0,
                     (struct sockaddr*)&your_adr, &adr_sz);
    total_len += str_len;
    printf("Message %d: %s \n", i+1, message);
}
```

버퍼에 쌓여 있다 한번에

- bound_host2.c 데이터 전송 부분

```
1) sendto(sock, msg1, sizeof(msg1), 0,
          (struct sockaddr*)&your_adr, sizeof(your_adr));

2) sendto(sock, msg2, sizeof(msg2), 0,
          (struct sockaddr*)&your_adr, sizeof(your_adr));

3) sendto(sock, msg3, sizeof(msg3), 0,
          (struct sockaddr*)&your_adr, sizeof(your_adr));
```

- 데이터 경계가 존재하기 때문에 5초간의 delay를 추가해도 총 3개의 메시지를 3번의 recvfrom()함수 호출을 통해 수신함

- 3번의 sendto() 함수 호출로 3개의 데이터를 전송
- 전달할 목적지 정보를 sendto() 함수 호출마다 지정

bound_host1.c (수신측) #1

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

#define BUF_SIZE 30
void error_handling(char *message);

int main(int argc, char *argv[])
{
    int sock;
    char message[BUF_SIZE];
    struct sockaddr_in my_adr, your_adr;
    socklen_t adr_sz;
    int str_len, i;
    int total_len = 0;

    if(argc!=2){
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }
    sock=socket(PF_INET, SOCK_DGRAM, 0);

    if(sock==-1)
        error_handling("socket() error");
```

bound_host1.c (수신측) #2

```
    memset(&my_adr, 0, sizeof(my_adr));
    my_adr.sin_family=AF_INET;
    my_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    my_adr.sin_port=htons(atoi(argv[1]));

    if(bind(sock, (struct sockaddr*)&my_adr, sizeof(my_adr))==-1)
        error_handling("bind() error");
    for(i=0; i<3; i++)
    {
        sleep(5); // delay 5 sec.
        adr_sz=sizeof(your_adr);
        str_len=recvfrom(sock, message, BUF_SIZE, 0,
                         (struct sockaddr*)&your_adr, &adr_sz);
        total_len += str_len;
        printf("Message %d: %s \n", i+1, message);
    }
    printf("Total received bytes: %d\n", total_len);
    close(sock);
    return 0;
}

void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

5초 간격으로
3번 recvfrom() 함수 호출

bound_host2.c (송신측) #1

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

#define BUF_SIZE 30
void error_handling(char *message);

int main(int argc, char *argv[])
{
    int sock;
    char msg1[]="Hi!";
    char msg2[]="I'm another UDP host!";
    char msg3[]="Nice to meet you";

    int len=0, total_len=0;
    struct sockaddr_in your_adr;
    socklen_t your_adr_sz;
    if(argc!=3){
        printf("Usage : %s <IP> <port>\n", argv[0]);
        exit(1);
    }
    sock=socket(PF_INET, SOCK_DGRAM, 0);
    if(sock==-1)
        error_handling("socket() error");
```

세 개의 메시지 전송
(msg1, msg2, msg3)

bound_host2.c (송신측) #2

```
    memset(&your_adr, 0, sizeof(your_adr));

    your_adr.sin_family=AF_INET;
    your_adr.sin_addr.s_addr=inet_addr(argv[1]);
    your_adr.sin_port=htons(atoi(argv[2]));

    len = sendto(sock, msg1, sizeof(msg1), 0, (struct sockaddr*)&your_adr, sizeof(your_adr));
    printf("sent bytes: %d\n", len);
    total_len += len;

    len = sendto(sock, msg2, sizeof(msg2), 0, (struct sockaddr*)&your_adr, sizeof(your_adr));
    printf("sent bytes: %d\n", len);
    total_len += len;

    len = sendto(sock, msg3, sizeof(msg3), 0, (struct sockaddr*)&your_adr, sizeof(your_adr));
    printf("sent bytes: %d\n", len);

    total_len += len;
    printf("Total sent bytes: %d\n", total_len);
    close(sock);
    return 0;
}
void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

실행 화면 (bound_host1, 2 통신)

bound_host1 실행 서버

```
$ gcc bound_host1.c -o bound_host1
$ ./bound_host1 9190
Message 1: Hi!
Message 2: I'm another UDP host!
Message 3: Nice to meet you
Total received bytes: 43
```

3번의 recvfrom() 호출

bound_host2 실행

```
$ gcc bound_host2.c -o bound_host2
$ ./bound_host2 127.0.0.1 9190
sent bytes: 4
sent bytes: 22
sent bytes: 17
Total sent bytes: 43
```

Connected UDP 소켓

- 기존 UDP 소켓의 sendto() 함수 호출 과정

  - 1단계: UDP 소켓에 목적지 IP와 Port 번호 등록
  - 2단계: 데이터 전송
  - 3단계: UDP 소켓에 등록된 목적지 정보 삭제

sendto()함수를 호출할 때마다 3단계를 반복

- Connected UDP 소켓
  - TCP와 같이 상대 소켓과의 연결을 의미하지는 않음
  - 1단계와 3단계의 과정을 거치지 않음
  - 소켓에 목적지 정보는 등록됨
  - read(), write() 함수 호출이 가능
  - 하나의 호스트와 장시간 데이터를 송수신
    - Connected UDP가 효율적임

이점을 유용하다.

UDP 소켓을 대상으로 connect() 함수 호출

Connected UDP 소켓 생성 과정

```
sock=socket(PF_INET, SOCK_DGRAM, 0);
if(sock==-1)
    error_handling("socket() error");

memset(&serv_adr, 0, sizeof(serv_adr));
serv_adr.sin_family=AF_INET;
serv_adr.sin_addr.s_addr=inet_addr(argv[1]);
serv_adr.sin_port=htons(atoi(argv[2]));

connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr));
```

uecho_con_client.c #1

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

#define BUF_SIZE 30
void error_handling(char *message);

int main(int argc, char *argv[])
{
    int sock;
    char message[BUF_SIZE];
    int str_len;
    socklen_t adr_sz;
    struct sockaddr_in serv_adr, from_adr;
    if(argc!=3){
        printf("Usage : %s <IP> <port>\n", argv[0]);
        exit(1);
    }
    sock=socket(PF_INET, SOCK_DGRAM, 0);
    if(sock==-1)
        error_handling("socket() error");

    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=inet_addr(argv[1]);
    serv_adr.sin_port=htons(atoi(argv[2]));
```

uecho_con_client.c #2

```
    connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr));

    while(1)
    {
        fputs("Insert message(q to quit): ", stdout);
        fgets(message, sizeof(message), stdin);
        if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
            break;
        /*
        sendto(sock, message, strlen(message), 0,
                (struct sockaddr*)&serv_adr, sizeof(serv_adr));
        */
        write(sock, message, strlen(message));

        /*
        adr_sz=sizeof(from_adr);
        str_len=recvfrom(sock, message, BUF_SIZE, 0,
                            (struct sockaddr*)&from_adr, &adr_sz);
        */
        str_len=read(sock, message, sizeof(message)-1);

        message[str_len]=0;
        printf("Message from server: %s", message);
    }
    close(sock);
    return 0;
}
```

connected UDP
- connect() 함수 호출

connected UDP
- sendto(), recvfrom 대신에 write(), read() 함수 사용

uecho_con_client.c #3

```c
void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

- uecho_server.c 서버 실행 화면

```
$ ./userver 9190
recvfrom() len=6
recvfrom() len=18
recvfrom() len=16
recvfrom() len=13
```

- uecho_con_client.c 실행 화면

```
$ ./uecho_con_client 127.0.0.1 9190
Insert message(q to quit): Hello
Message from server: Hello
Insert message(q to quit): Nice to meet you.
Message from server: Nice to meet you.
Insert message(q to quit): Have a good day
Message from server: Have a good day
Insert message(q to quit): q
```

Questions?

LMS Q&A 게시판에 올려주세요.