

네트워크프로그래밍-3주

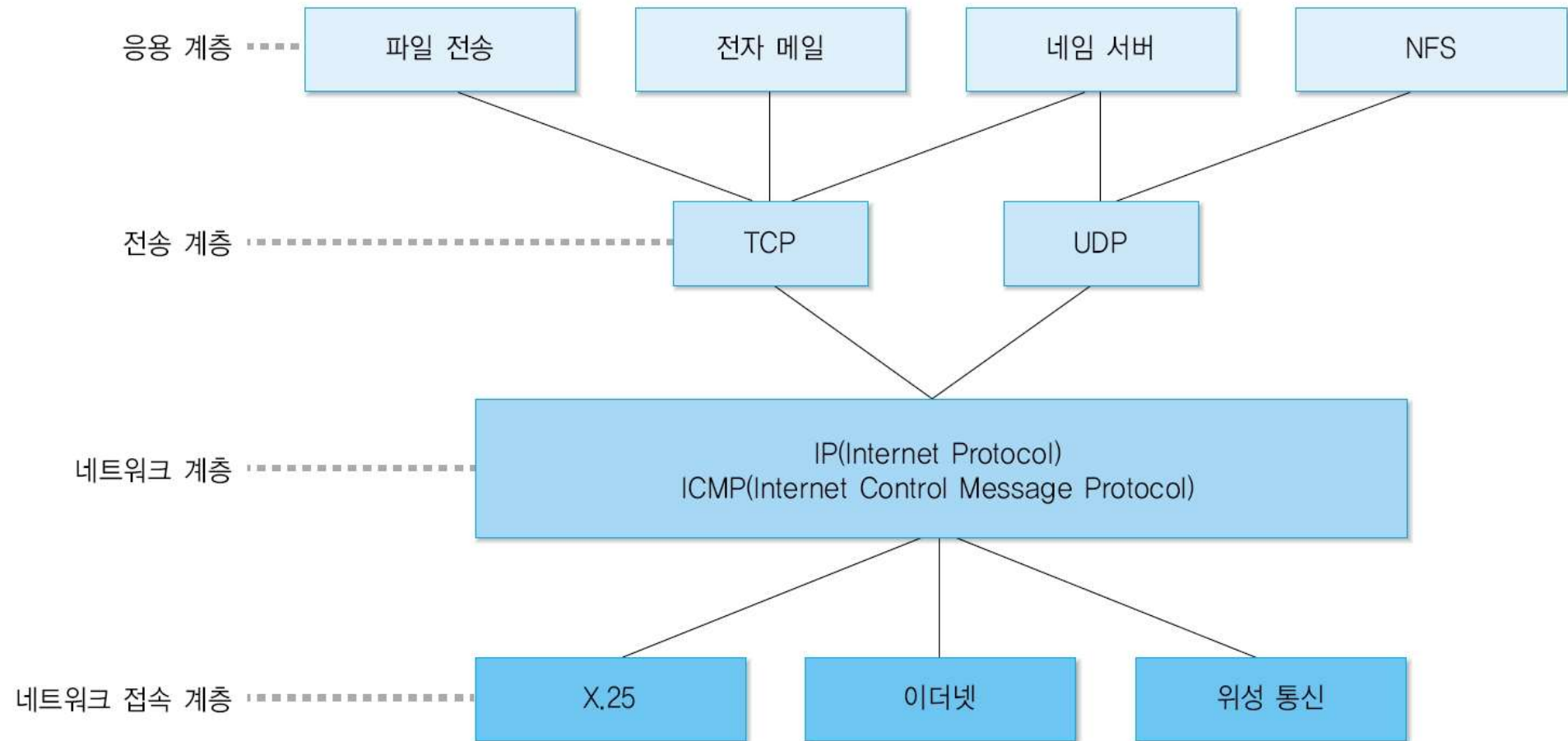
Socket 프로그래밍-1

정인환교수

Socket API, 소켓프로그래밍

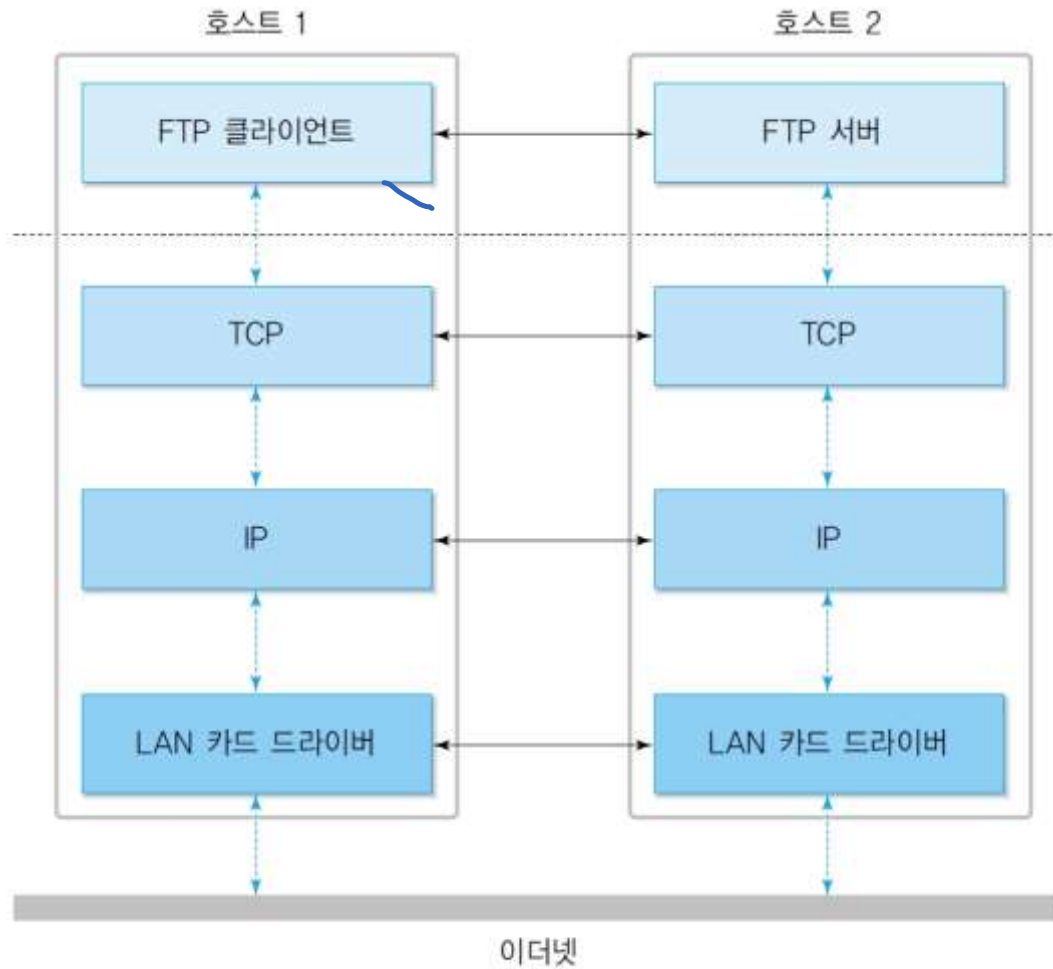
- ▶ TCP/IP 환경
- ▶ 소켓이란?
- ▶ Client/Server 통신 절차
- ▶ 소켓프로그래밍
- ▶ Socket API
- ▶ Socket API - 주소의 표현
- ▶ Socket API - 시스템콜 함수들
- ▶ 소켓프로그래밍 예1 - TCP Time Client/Server
- ▶ 소켓프로그래밍 예2 - UDP Time Client/Server
- ▶ 개발환경 구축
 - Vmware + Unbuntu Linux
- ▶ 과제설명 ✓

TCP/IP 구조

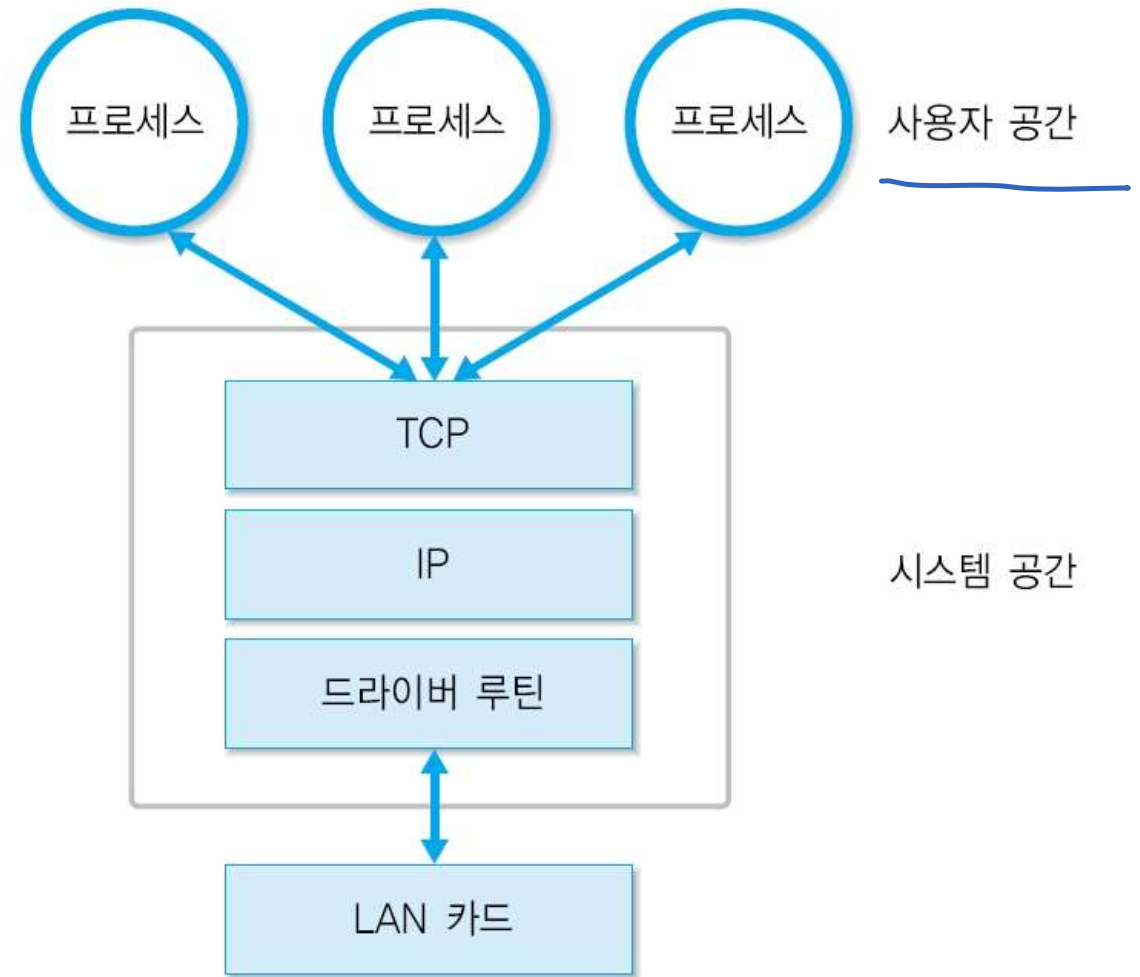


[그림 9-7] TCP/IP 구조

TCP/IP 환경



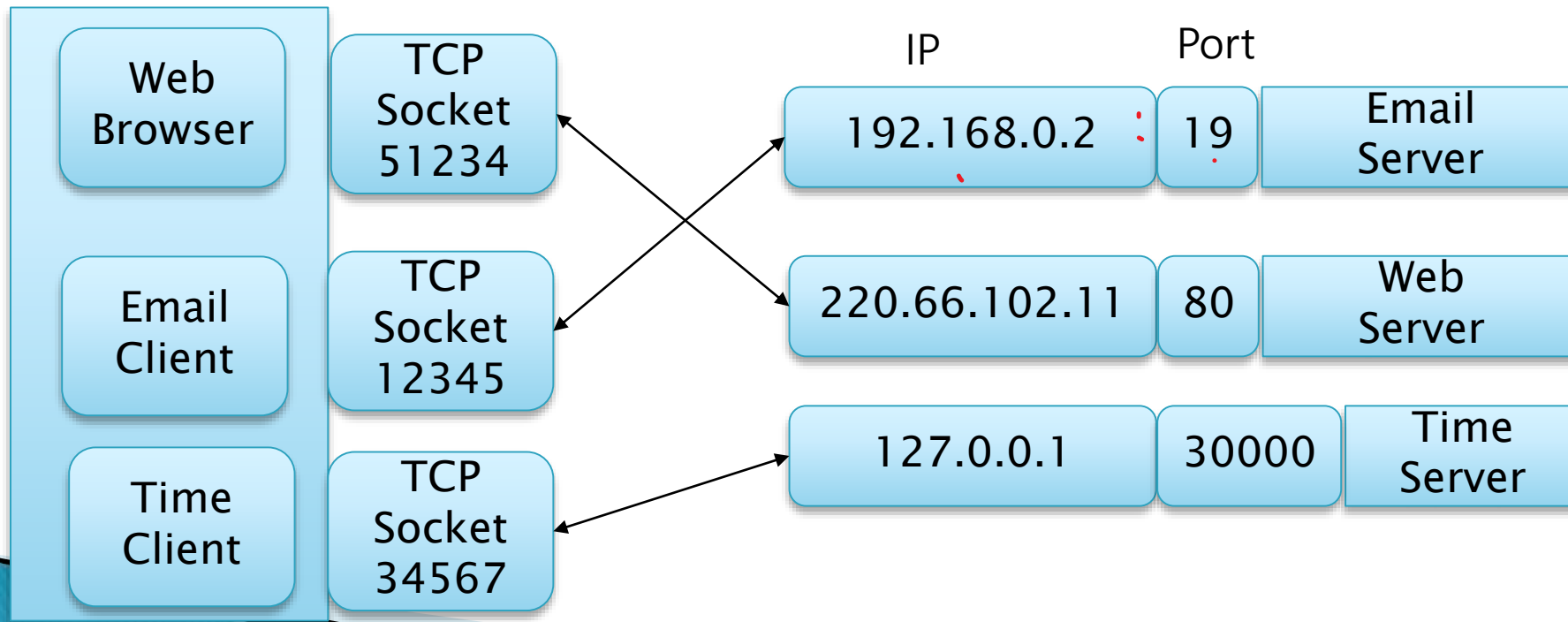
[그림 1-6] FTP의 계층 구조



[그림 2-8] TCP/IP 구현 환경

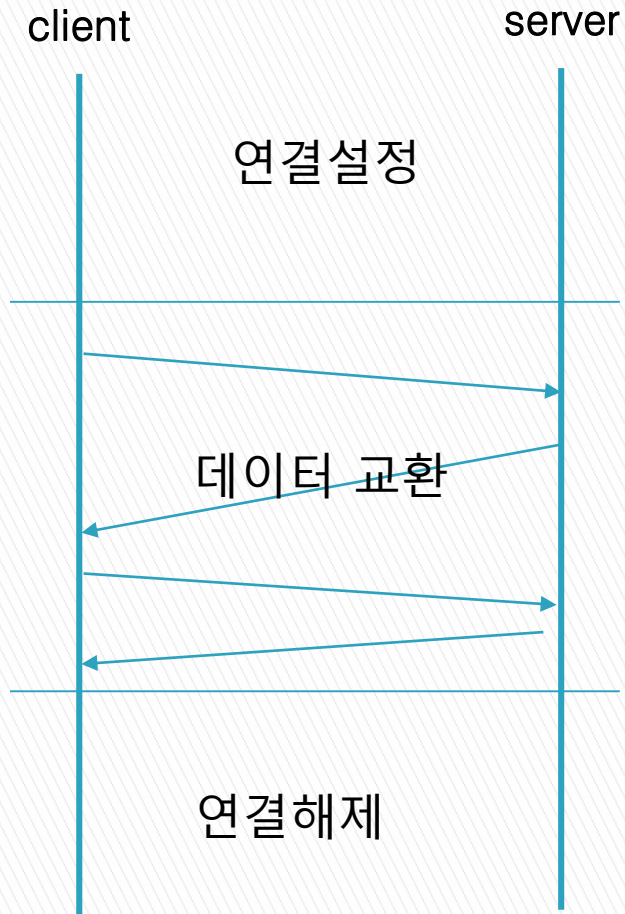
소켓이란?

- ▶ TCP/UDP IP 환경에서 통신을 위한 통로
 - send / recv 함수를 통해 Data 송/수신
 - IP + Port 번호로 구성
 - 예) 220.66.10.11 + 80 ==> 220.66.102.11:80

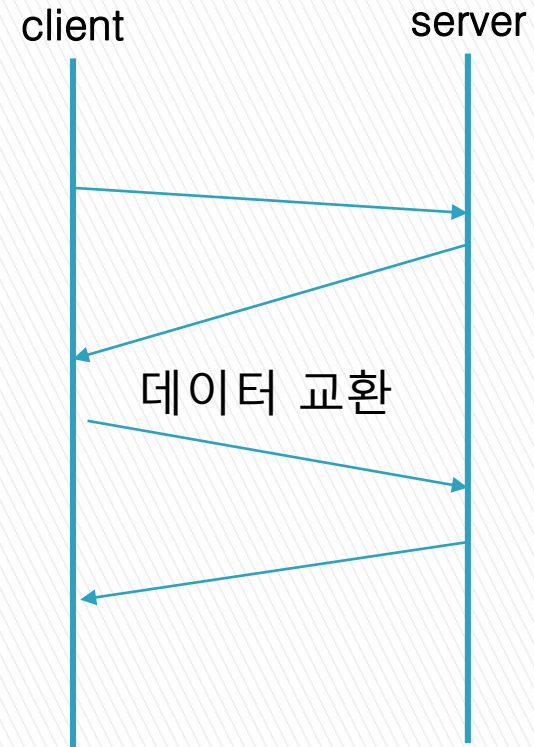


Client / Server 통신 절차

▶ TCP 통신 : 연결형



▶ UDP 통신 : 비 연결형



소켓프로그래밍

- ▶ Socket 함수
 - C언어 함수 들(socket, connect, accept, send, recv...)
 - 1982 버클리 대학(University of California at Berkeley)
 - BSD(Berkeley Software Distribution) UNIX 4.1
 - 1986년 BSD UNIX 4.3에서
 - BSD소켓 또는 버클리 소켓이라고 불림
- ▶ Windows 환경
 - WinSock
 - BSD Socket (Unix/linux)와 99% 호환
- ▶ Java 환경 (Socket Class)
 - 객체지향 방식
 - Socket Class
 - ServerSocket serverSocket;
 - Socket clientSocket;
- ▶ Python 환경
 - Java 환경과 유사
 - 객체지향 방식

Socket API

- ▶ Application Programming Interface
- ▶ 소켓 함수들

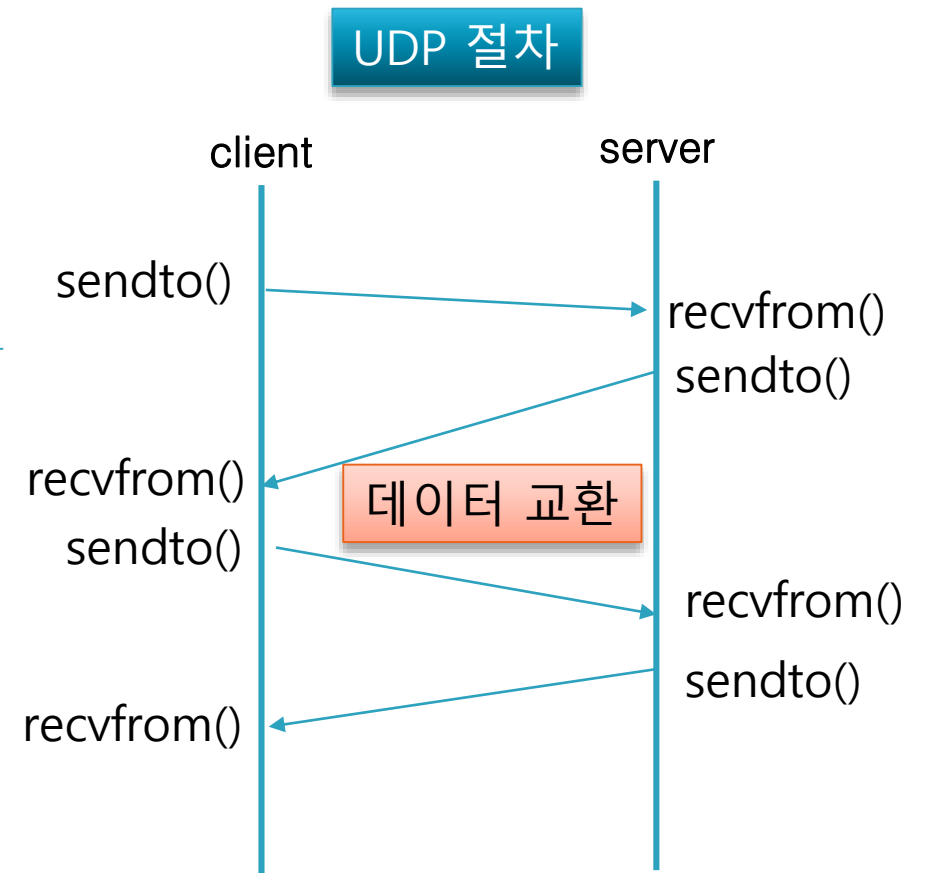
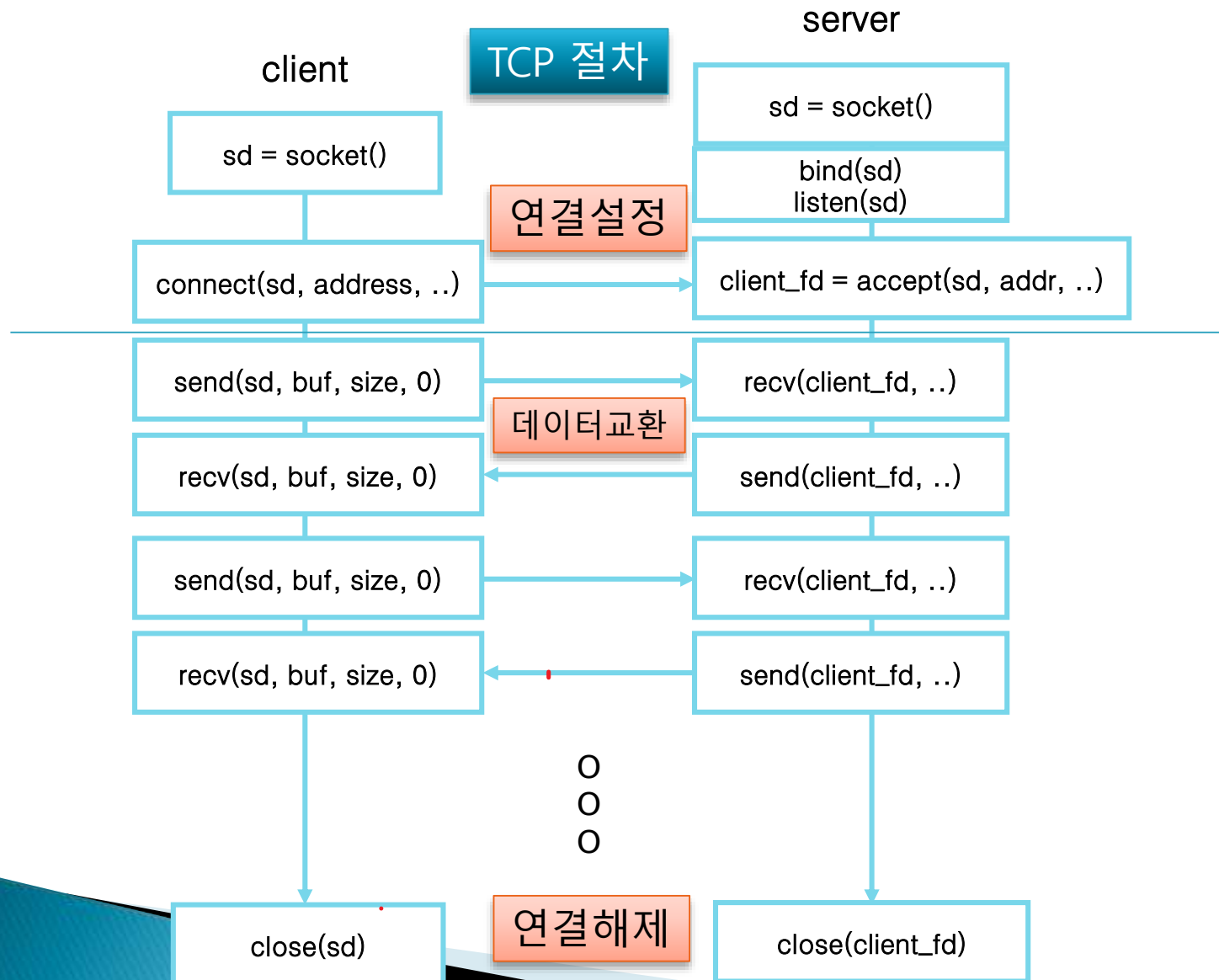
- ▶ TCP API

- socket
- connect (client)
- bind
- listen
- accept (server)
- send
- recv
- close
- inet_addr
- inet_ntos

- ▶ UDP API

- socket
- sendto
- recvfrom
- close

Socket API 흐름 : TCP / UDP



Socket API - 주소의 표현

▶ 소켓 주소

- 프로토콜의 종류에 따라 사용하는 주소 체계가 다름
- AF_UNIX: 한 호스트에 존재하는 프로세스 사이의 통신을 지원
- AF_INET: 다른 호스트에 존재하는 프로세스 사이의 통신을 지원

◦ 유닉스 주소 체계

- AF_UNIX
- 한 호스트에 존재하는 프로세스 사이의 통신을 지원
- 주소 체계는 파일 시스템의 경로명을 기반으로 함

- 주소 체계

```
struct sockaddr_un {  
    short sun_family;    /* AF_UNIX */  
    char sun_path[108]; /* pathname */  
};
```

Socket API - 주소의 표현

▶ 소켓 주소

◦ 인터넷 주소 체계

- AF_INET

- 다른 호스트에 존재하는 프로세스 사이의 통신을 지원
- 주소 체계는 32비트 IP 주소와 16 비트 포트 번호를 기반으로 함

- 주소 체계

```
struct sockaddr_in {  
    short sin_family;           /* AF_INET */  
    u_short sin_port;          /* Port Number */  
    struct in_addr sin_addr;   /* IP Address */  
    char sin_zero[8];         /* Padding */  
};  
  
struct in_addr {  
    u_long s_addr;  
};
```

Socket API - 주소의 표현

▶ 소켓 주소

◦ 통합 주소 체계

- 프로토콜마다 주소 체계를 지원하는 문법 구조가 다름
- 문법 구조상 하나의 함수에서 다양한 주소 체계를 지원하는데 어려움이 있음
- 따라서 모든 주소 체계를 수용할 수 있는 공통 주소 체계가 필요함

• 주소 체계

```
struct sockaddr {  
    u_short sa_family;      /* AF_UNIX, AF_INET, ... */  
    char sa_data[14];  
};
```

Socket API - 주소의 표현

▶ 소켓 주소

◦ 통합 주소 체계

• 사용 예

- addr: 주소 공간 자체는 해당 프로토콜의 주소 체계로 선언 (인터넷 주소 체계)
- bind() 함수의 두 번째 매개 변수는 문법적으로 공통 주소 체계만 수용

```
struct sockaddr_in addr;                /* 인터넷 주소 체계로 변수 선언 */
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY); /* IP 주소 */
addr.sin_port = htons(5010);            /* 포트 번호 */
bind(socket, (struct sockaddr *)&addr, sizeof(addr));
```

Socket API - 주소의 표현

▶ 소켓 서비스

◦ 소켓 유형

- SOCK_STREAM
 - 연결형 서비스를 지원
 - AF_INET에서는 TCP 프로토콜을 사용
- SOCK_DGRAM
 - 비연결형 서비스를 지원
 - AF_INET에서는 UDP 프로토콜을 사용
- SOCK_RAW
 - IP 프로토콜을 직접 사용

Socket API - 주소의 표현

▶ 소켓 서비스

◦ 소켓 함수

- `s = socket (int domain, int type, int protocol)`
 - 매개 변수로 지정된 유형을 지원하는 소켓을 생성
 - 생성된 소켓을 가리키는 파일 디스크립터를 리턴
- `bind (int s, struct sockaddr *name, socklen_t *namelen)`
 - `s`가 가리키는 소켓에 소켓 주소를 부여함
 - `name`: 소켓 주소
- `listen (int s, int backlog)`
 - 소켓을 활성화 시킴
- `accept (int s, struct sockaddr *addr, socklen_t *addrlen)`
 - 클라이언트/서버 환경에서 서버가 대기하는 역할을 함
 - 클라이언트의 `connect()` 함수와 만나면 소켓 연결을 설정함

Socket API - 주소의 표현

▶ 소켓 서비스

◦ 소켓 함수

- `connect (int s, struct sockaddr *name, socklen_t namelen)`
 - 클라이언트/서버 환경에서 클라이언트의 연결 설정 요청을 수행함
 - 서버의 `accept()` 함수와 만나면 소켓 연결을 설정함
- `send (int s, void *msg, size_t len, int flags)`
 - 연결이 설정된 소켓에 데이터를 송신
 - 전송 데이터는 `msg`가 가리킴
- `recv (int s, void *buf, size_t len, int flags)`
 - 연결이 설정된 소켓에서 데이터를 수신
 - 수신 데이터는 `buf`가 가리키는 공간에 저장됨

Socket API - 시스템 콜

▶ socket() 함수

- 소켓을 생성하며, 생성된 소켓의 디스크립터를 반환

◦ socket() 함수 사용법

- 문법

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

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

```
int socket(int domain, int type, int protocol);
```

- 설명

- domain: 사용할 도메인을 지정
- type: 서비스 유형을 지정
- protocol: 보통 0으로 지정

Socket API - 시스템 콜

▶ socket() 함수 예제

```
sd = socket(AF_UNIX, SOCK_STREAM, 0);  
/* 유닉스 도메인 연결형 서비스 */
```

```
sd = socket(AF_UNIX, SOCK_DGRAM, 0);  
/* 유닉스 도메인 비연결형 서비스 */
```

```
sd = socket(AF_INET, SOCK_STREAM, 0);  
/* 인터넷 도메인 연결형 서비스 */
```

```
sd = socket(AF_INET, SOCK_DGRAM, 0);  
/* 인터넷 도메인 비연결형 서비스 */
```

Socket API - 시스템 콜

▶ bind() 함수

- 생성된 소켓에 주소를 부여

◦ bind() 함수 사용법

- 문법

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

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

```
int bind(int s, const struct sockaddr *name, socklen_t *namelen);
```

- 설명

- s: socket() 함수가 리턴한 디스크립터
- name: 바인드할 소켓 주소를 표기
- namelen: name에 보관된 주소 공간의 크기

Socket API - 시스템 콜

- ▶ bind() 함수
 - AF_UNIX 예제

```
int sd;
struct sockaddr_un addr;

sd = socket(AF_UNIX, SOCK_STREAM, 0);
if(sd == -1) {
    perror("socket");
    exit(1);
}

addr.sun_family = AF_UNIX;
strcpy(addr.sun_path, "/tmp/sock_addr");

if(bind(sd, (struct sockaddr *)&addr, sizeof(addr)) == -1) {
    perror("bind");
    exit(1);
}
```

Socket API - 시스템 콜

- ▶ bind() 함수
 - AF_INET 예제

```
int sd;
struct sockaddr_in addr;

sd = socket(AF_INET, SOCK_STREAM, 0);
if(sd == -1) {
    perror("socket");
    exit(1);
}

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(5010);

if(bind(sd, (struct sockaddr *)&addr, sizeof(addr)) == -1) {
    perror("bind");
    exit(1);
}
```

Socket API - 시스템 콜

▶ bind() 함수

◦ 주소 변환

- 컴퓨터 마다 정수형 데이터를 처리하는 방법이 다를 수 있음
- '개별 호스트 -> 네트워크' 변환: htonl(), htons()
- '네트워크 -> 개별 호스트' 변환: ntohl(), ntohs()

• 문법

```
#include <sys/types.h>
#include <netinet/in.h>
#include <inttypes.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

Socket API - 시스템 콜

▶ listen() 함수

- 소켓에서 대기할 수 있는 연결 요청의 개수를 지정

◦ listen() 함수 사용법

- 문법

```
#include<sys/types.h>
#include<sys/socket.h>
int listen(int s, int backlog);
```

- 설명

- s: socket() 함수가 생성한 연결형 서비스용 소켓
- backlog: 일반적인 환경에서 5로 지정

Socket API - 시스템 콜

▶ listen() 함수

```
int sd;  
struct sockaddr_in addr;
```

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

```
addr.sin_family = AF_INET;  
addr.sin_addr.s_addr = htonl (INADDR_ANY);  
addr.sin_port = htons (5010);
```

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

```
if (listen (sd, 5) == -1) {  
    perror ("listen");  
    exit (1);  
}
```


Socket API - 시스템 콜

▶ accept() 함수

- 서버 프로그램에서 클라이언트의 연결 요청을 대기

◦ accept() 함수 사용법

- 문법

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

- 설명

- s: socket() 함수가 생성한 연결형 서비스용 소켓
- addr: 연결을 요청한 클라이언트의 소켓 주소를 반환

Socket API - 시스템 콜

▶ accept() 함수

```
int sd, new;
struct sockaddr_in addr;
struct sockaddr_in client;
int client_len;

sd = socket (AF_INET, SOCK_STREAM, 0);
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl (INADDR_ANY);
addr.sin_port = htons (5010);
bind (sd, (struct sockaddr *)&addr, sizeof (addr));
listen (sd, 5);

while ((new = accept (sd, (struct sockaddr *)&client, &client_len)) != -1) {
    if (fork() == 0) {                /* 자식 프로세스 */
        close (sd);
        work (new);                  /* new를 이용해 클라이언트와 통신 */
        close (new);
        exit (0);
    }
    close (new) /* 부모 프로세스 */
}
```

Socket API - 시스템 콜

▶ connect() 함수

- 클라이언트 프로그램에서 서버에게 연결 요청을 수행

◦ connect() 함수 사용법

- 문법

```
#include<sys/types.h>
#include<sys/socket.h>
int connect(int s, const struct sockaddr *name, socklen_t namelen);
```

- 설명

- s: socket() 함수가 생성한 연결형 서비스용 소켓
- name: 연결하고자 하는 서버의 소켓 주소

Socket API - 시스템 콜

▶ connect() 함수 예

```
#define TIME_SERVER      "127.0.0.1"  
#define TIME_PORT        30000
```

```
main (  
{
```

```
    int sock;  
    struct sockaddr_in server;  
    char buf [256];
```

```
    sock = socket (AF_INET, SOCK_STREAM, 0);
```

```
    server.sin_family = AF_INET;  
    server.sin_addr.s_addr = htonl (inet_addr (TIME_SERVER));  
    server.sin_port = htons (TIME_PORT);
```

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

Socket API - 시스템 콜

▶ connect() 함수

◦ 주소 변환

- IP 주소의 표기 방식

- 10진수 표기 방식: 사람들의 편의를 위하여 211.223.201.30 등의 형식을 사용
- 2진수 표기 방식: IP 프로토콜에서 사용

- inet_addr(): 10진수 형식을 2진수 형식으로 변환

- inet_ntoa(): 2진수 형식을 10진수 형식으로 변환

- 문법

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
unsigned long inet_addr(const char *cp);
char *inet_ntoa(const struct in_addr in);
```

Socket API - 시스템 콜

▶ send() 함수

- send(): 연결형 서비스에서 데이터를 송신
- sendto(): 비연결형 서비스에서 데이터를 송신

◦ send() 함수 사용법

- 문법

```
#include<sys/types.h>
#include<sys/socket.h>
ssize_t send(int s, const void *msg, size_t len, int flags);
ssize_t sendto(int s, const void *msg, size_t len, int flags,
               const struct sockaddr *to, socklen_t tolen);
```

• 설명

- s: socket() 함수가 생성한 소켓
- msg: 송신할 데이터
- to: 비연결형 서비스에서 수신자 주소

Socket API - 시스템 콜

▶ send() 함수 예제

```
int sd;  
struct sockaddr_in addr;  
char *data = "Test Message";  
int length = strlen (data) + 1;  
  
sd = socket (AF_INET, SOCK_STREAM, 0);  
addr.sin_family = AF_INET;  
addr.sin_addr.s_addr = htonl (inet_addr ("211.223.201.30"));  
addr.sin_port = htons (5010);  
connect (sd, (struct sockaddr *)&addr, sizeof (addr));
```

```
if (send (sd, data, length, 0) == -1) {  
    perror ("send");  
    exit (1);  
}
```

Socket API - 시스템 콜

▶ recv() 함수

- recv(): 연결형 서비스에서 데이터를 수신
- recvfrom() : 비연결형 서비스에서 데이터를 수신

◦ recv() 함수 사용법

- 문법

```
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/uio.h>
ssize_t recv(int s, void *buf, size_t len, int flags);
ssize_t recvfrom(int s, void *buf, size_t len, int flags,
                 struct sockaddr *from, socklen_t *fromlen);
```

- 설명

- s: socket() 함수가 생성한 소켓
- buf: 수신할 데이터를 저장할 공간
- from: 비연결형 서비스에서 송신자 주소

Socket API - 시스템 콜

▶ recv() 함수 예제

```
#define TIME_SERVER    "127.0.0.1"
#define TIME_PORT      30000

main ()
{
    int sock;
    struct sockaddr_in server;
    char buf [256];

    sock = socket (AF_INET, SOCK_STREAM, 0);

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl (inet_addr (TIME_SERVER));
    server.sin_port = htons (TIME_PORT);

    connect (sock, (struct sockaddr *)&server, sizeof(server));

    if (recv (sock, buf, sizeof (buf), 0) == -1)
        exit (1);
    printf ("Time information from server is %s", buf);
    close (sock);
}
```

소켓프로그래밍 예1 - Time Client / Server

- ▶ Client 가 요청하면 Server가 현재 시간을 문자열로 Return
- ▶ Linux
 - NetP03-linux.zip
 - Makefile, time_client.c, time_server.c, udp_time_client.c, udp_time_server.c
 - make 명령어로 compile
- ▶ Windows
 - NetP03-Win.zip
 - NetP03-Win.sln 으로 시작
 - wtime_client, wtime_server, wudp_time_client, wudp_time_server
 - Linux 와 같은 방법으로 실행 또는
 - IP, Port 번호 변경
 - 프로젝트 > 속성 > 구성속성 > 디버깅 > 명령인수 설정

Time 함수 사용 예

// wctime.c : ctime() 값 출력

```
#include <stdio.h>
```

```
#include <time.h>
```

```
#include <string.h>
```

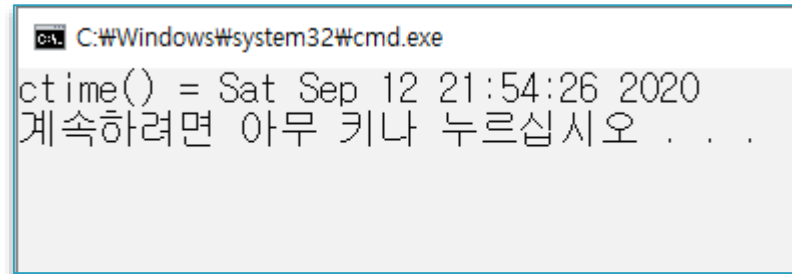
```
void main() {
```

```
    time_t today;
```

```
    time(&today); // today = time(NULL);
```

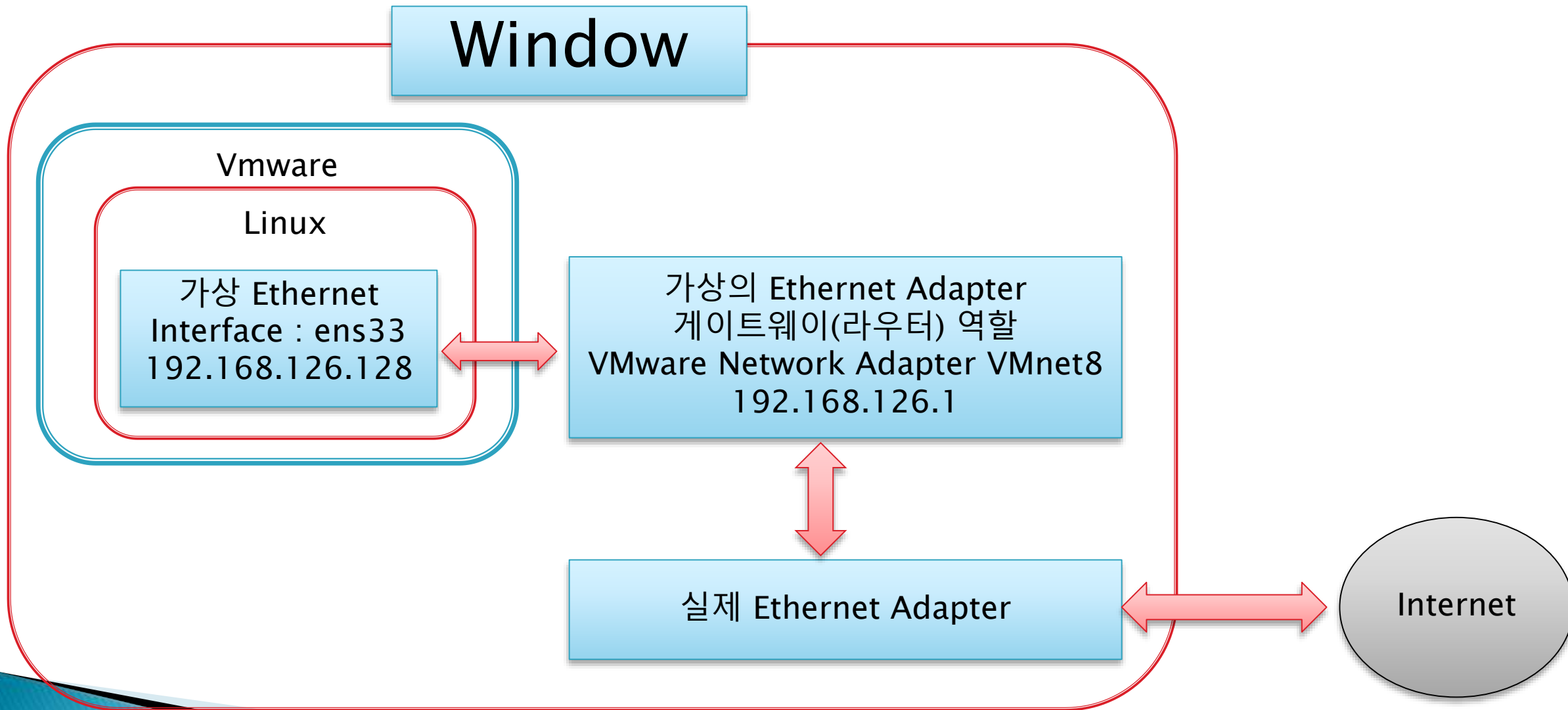
```
    printf("ctime() = %s", ctime(&today));
```

```
}
```



A screenshot of a Windows command prompt window. The title bar reads "C:\Windows\system32\cmd.exe". The command prompt shows the output of the `ctime()` function: `ctime() = Sat Sep 12 21:54:26 2020`. Below this, there is a line of Korean text: `계속하려면 아무 키나 누르십시오 . . .`.

Linux / Windows Network 환경



소켓프로그래밍 예1 - TCP time client/server

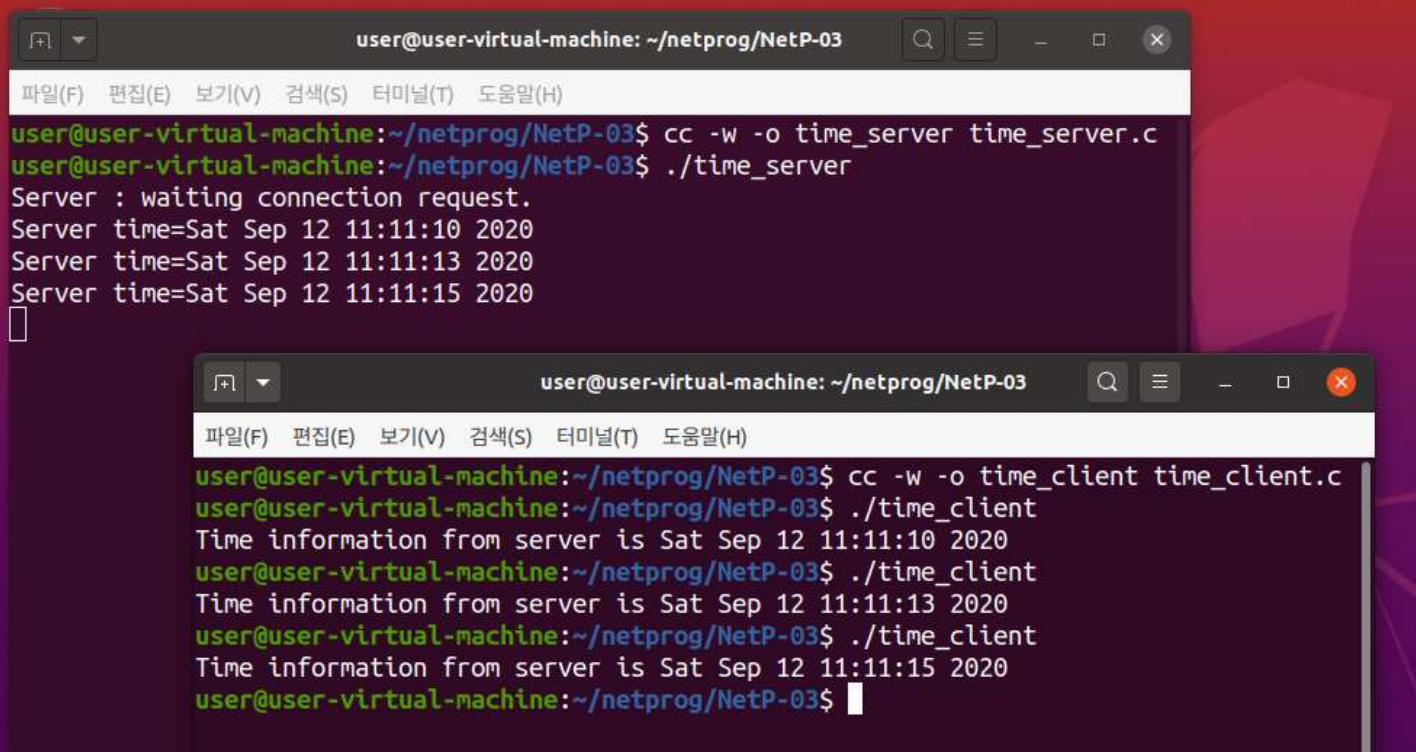
- ▶ Server가 현재 시간을 문자열로 Return

- ▶ time_client/server

- `cc -w -o time_client time_client.c`
- `cc -w -o time_server time_server.c`
- make 명령어 사용

- ▶ 실행 방법

- `./time_server` 먼저
- `./time_client [IP] [Port]`

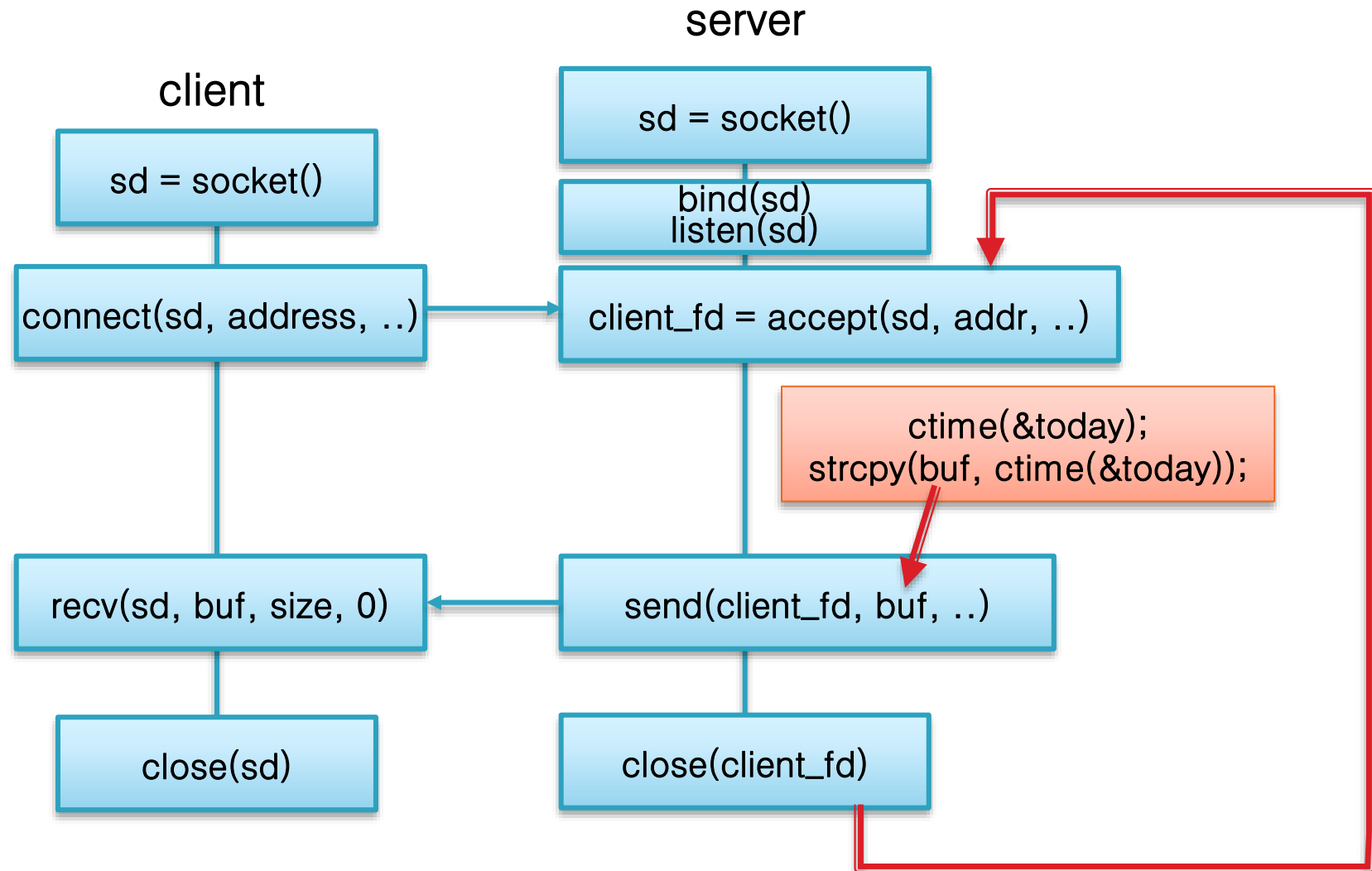


The image shows two terminal windows from a user@user-virtual-machine. The top window shows the compilation and execution of the time_server program. The bottom window shows the compilation and execution of the time_client program, which receives time information from the server.

```
user@user-virtual-machine: ~/netprog/NetP-03
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
user@user-virtual-machine:~/netprog/NetP-03$ cc -w -o time_server time_server.c
user@user-virtual-machine:~/netprog/NetP-03$ ./time_server
Server : waiting connection request.
Server time=Sat Sep 12 11:11:10 2020
Server time=Sat Sep 12 11:11:13 2020
Server time=Sat Sep 12 11:11:15 2020
^

user@user-virtual-machine: ~/netprog/NetP-03
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
user@user-virtual-machine:~/netprog/NetP-03$ cc -w -o time_client time_client.c
user@user-virtual-machine:~/netprog/NetP-03$ ./time_client
Time information from server is Sat Sep 12 11:11:10 2020
user@user-virtual-machine:~/netprog/NetP-03$ ./time_client
Time information from server is Sat Sep 12 11:11:13 2020
user@user-virtual-machine:~/netprog/NetP-03$ ./time_client
Time information from server is Sat Sep 12 11:11:15 2020
user@user-virtual-machine:~/netprog/NetP-03$
```

소켓프로그래밍 예1 - TCP time client/server



소켓프로그래밍 예1 - TCP time client/server

▶ time_server.c

```
# define TIME_PORT    30000
main (int argc, char *argv[])
{
    int sock, sock2;
    struct sockaddr_in server, client;
    int len;
    char buf [256];
    time_t today;

    sock = socket (AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl (INADDR_ANY);
    server.sin_port = htons (TIME_PORT);
    bind (sock, (struct sockaddr *)&server, sizeof (server));
    listen (sock, 5);
    while (1) {
        sock2 = accept (sock, (struct sockaddr *)&client, &len);
        time (&today);
        strcpy (buf, ctime (&today));
        send (sock2, buf, strlen (buf) + 1, 0);
        close (sock2);
    }
}
```

소켓프로그래밍 예1 - TCP time client/server

▶ time_client.c

```
#define TIME_SERVER    "127.0.0.1"
#define TIME_PORT      30000

void main(int argc, char *argv[]) {
    int sock;
    struct sockaddr_in server;
    char *haddr;
    char buf[BUF_LEN+1] = {0};
    char *ip_addr = TIME_SERVER, *port_no = TIME_PORT;

    sock = socket(AF_INET, SOCK_STREAM, 0);

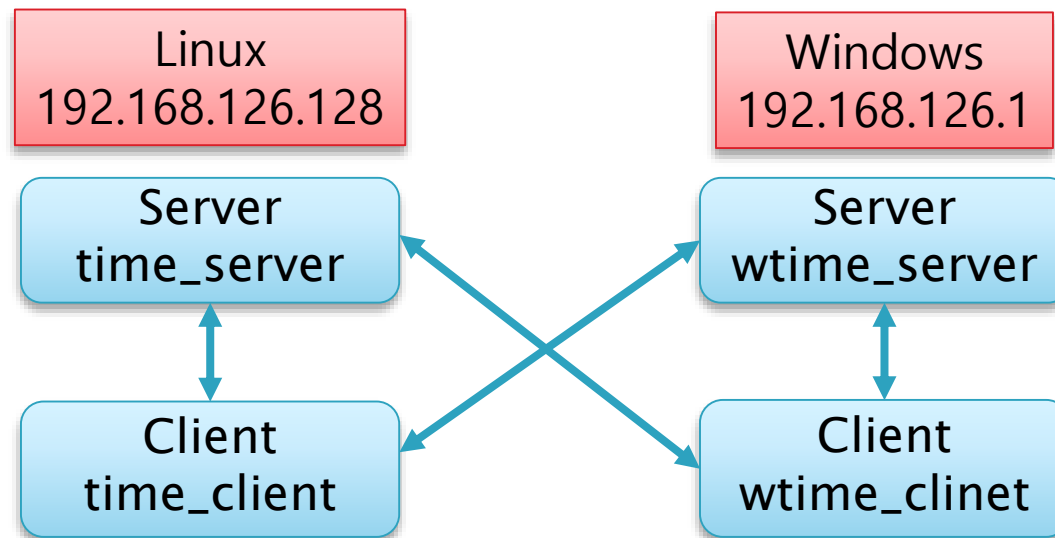
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(inet_addr (ip_addr));
    server.sin_port = htons(atoi(port_no));

    connect(sock, (struct sockaddr *)&server, sizeof(server));

    if (recv(sock, buf, sizeof (buf), 0) == -1)
        exit (1);
    printf ("Time information from server is %s", buf);
    close (sock);
}
```

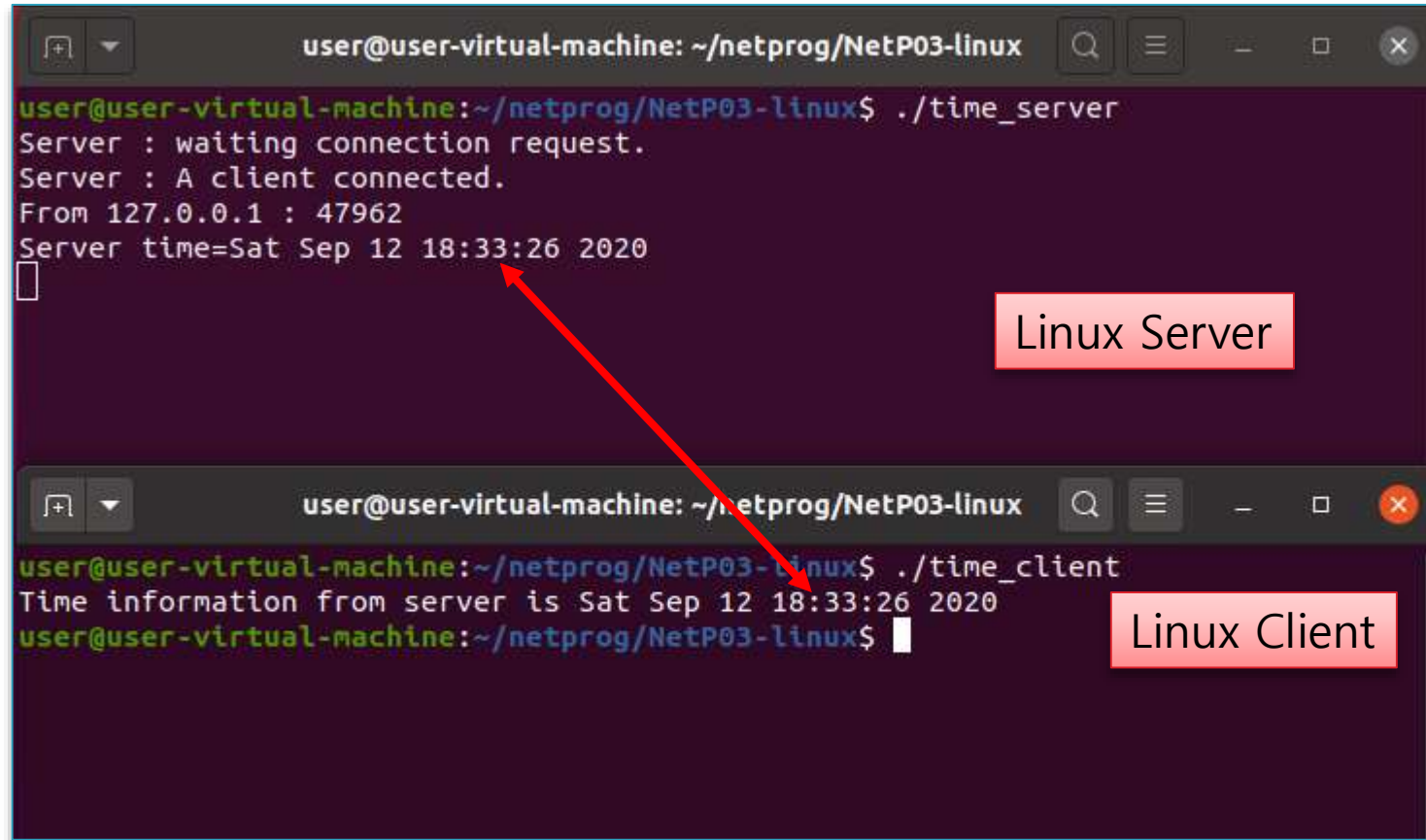

소켓프로그래밍 예1 - TCP time client/server

- ▶ Linux Client / Linux Server
- ▶ Windows Client / Windows Server
- ▶ Windows Client / Linux Server
- ▶ Linux Client / Windows Server



TCP time client/server 실행 화면

▶ Linux Client / Linux Server



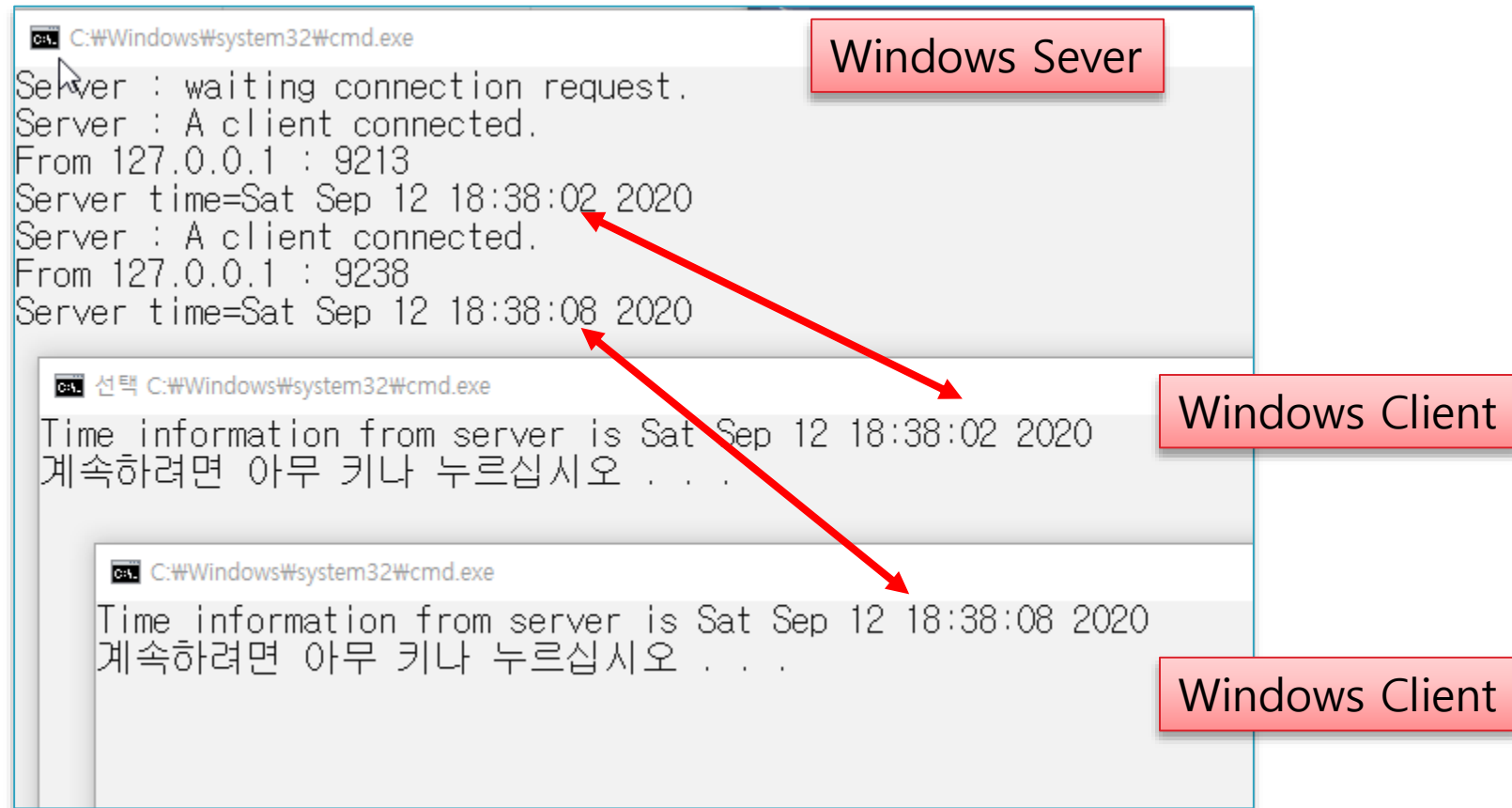
The image shows two terminal windows from a user@user-virtual-machine. The top window is running the `./time_server` program, which outputs: "Server : waiting connection request.", "Server : A client connected.", "From 127.0.0.1 : 47962", and "Server time=Sat Sep 12 18:33:26 2020". The bottom window is running the `./time_client` program, which outputs: "Time information from server is Sat Sep 12 18:33:26 2020". A red arrow points from the time output in the client window to the time output in the server window. Labels "Linux Server" and "Linux Client" are placed next to their respective windows.

```
user@user-virtual-machine: ~/netprog/NetP03-linux
user@user-virtual-machine:~/netprog/NetP03-linux$ ./time_server
Server : waiting connection request.
Server : A client connected.
From 127.0.0.1 : 47962
Server time=Sat Sep 12 18:33:26 2020
█

user@user-virtual-machine: ~/netprog/NetP03-linux
user@user-virtual-machine:~/netprog/NetP03-linux$ ./time_client
Time information from server is Sat Sep 12 18:33:26 2020
user@user-virtual-machine:~/netprog/NetP03-linux$ █
```

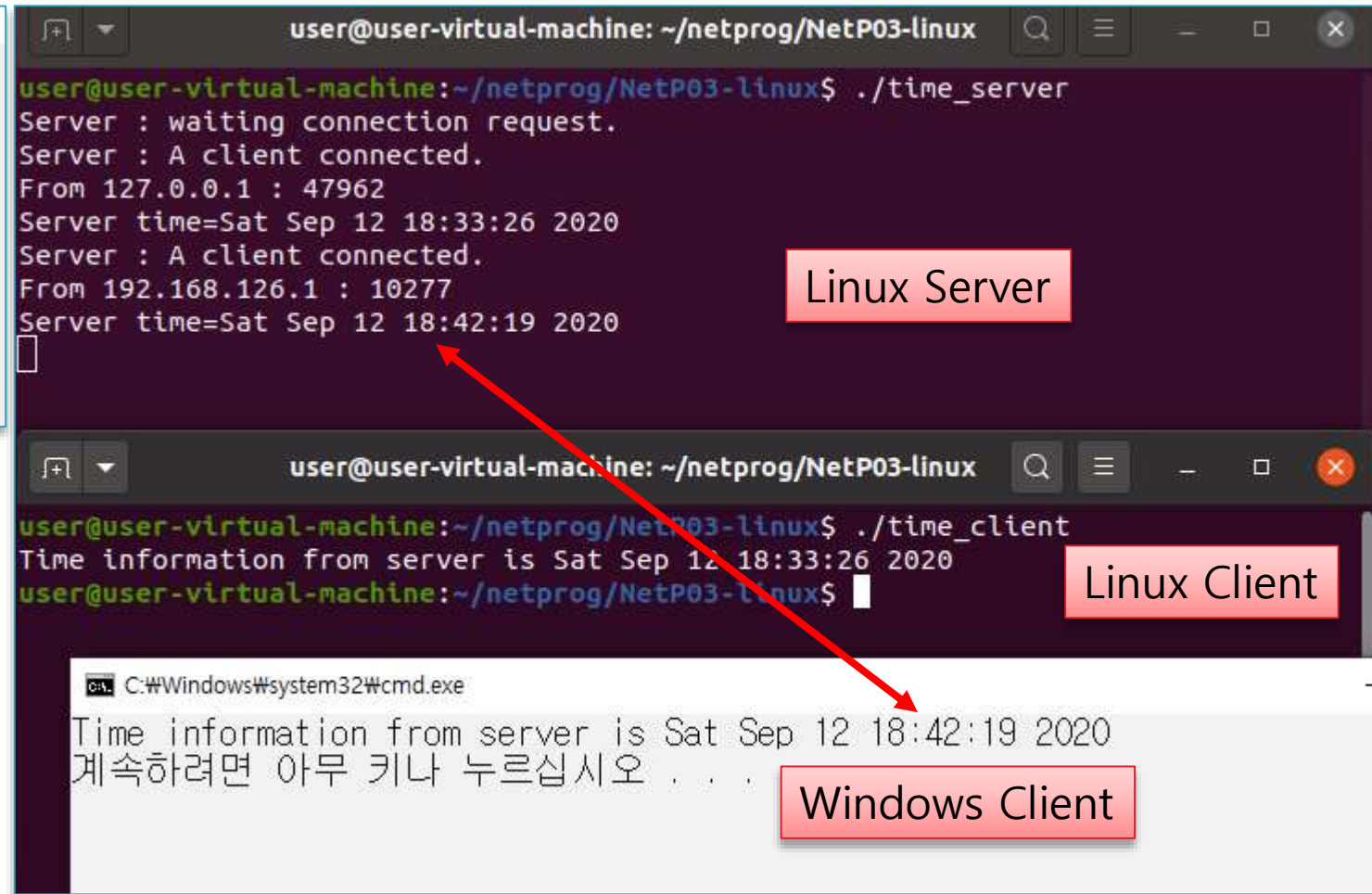
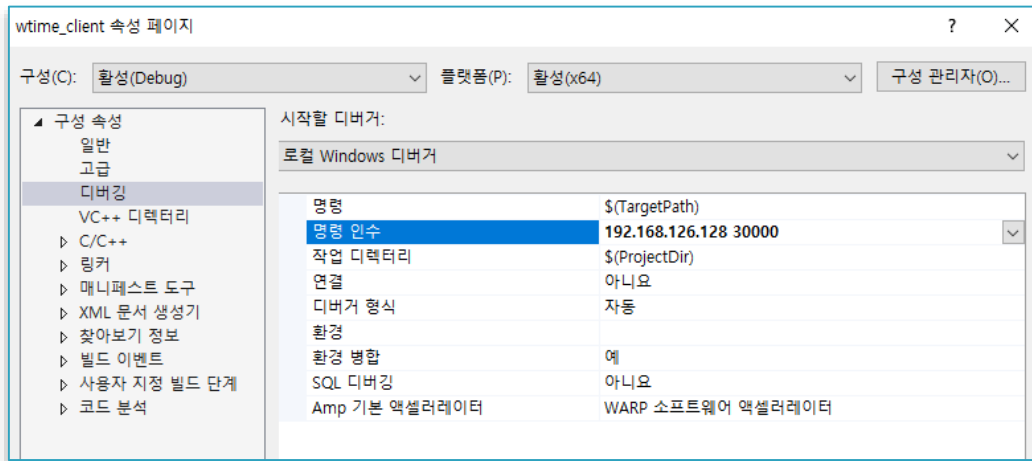
TCP time client/server 실행 화면

▶ Windows Client / Windows Server



TCP time client/server 실행 화면

▶ Windows Client > Linux Server



TCP time client/server 실행 화면

▶ Linux Client / Windows Server

The screenshot displays a virtual machine environment with three terminal windows. The top window, titled 'Linux Server', shows the execution of the `./time_server` command. It receives two connection requests: one from `127.0.0.1 : 47962` and another from `192.168.126.1 : 10277`. The bottom window, titled 'Linux Client', shows the execution of the `./time_client` command, which receives time information from the server: 'Time information from server is Sat Sep 12 18:33:26 2020'. The middle window, titled 'Windows Server', shows the execution of the `cmd.exe` command, which receives two connection requests: one from `127.0.0.1 : 11576` and another from `192.168.126.128 : 50406`. The bottom window, titled 'Windows Client', shows the execution of the `cmd.exe` command, which receives time information from the server: 'Time information from server is Sat Sep 12 18:47:34 2020'. Red arrows indicate the flow of data from the Linux Server to the Windows Server and from the Linux Client to the Windows Client. Blue arrows indicate the flow of data from the Linux Server to the Linux Client and from the Windows Server to the Windows Client.

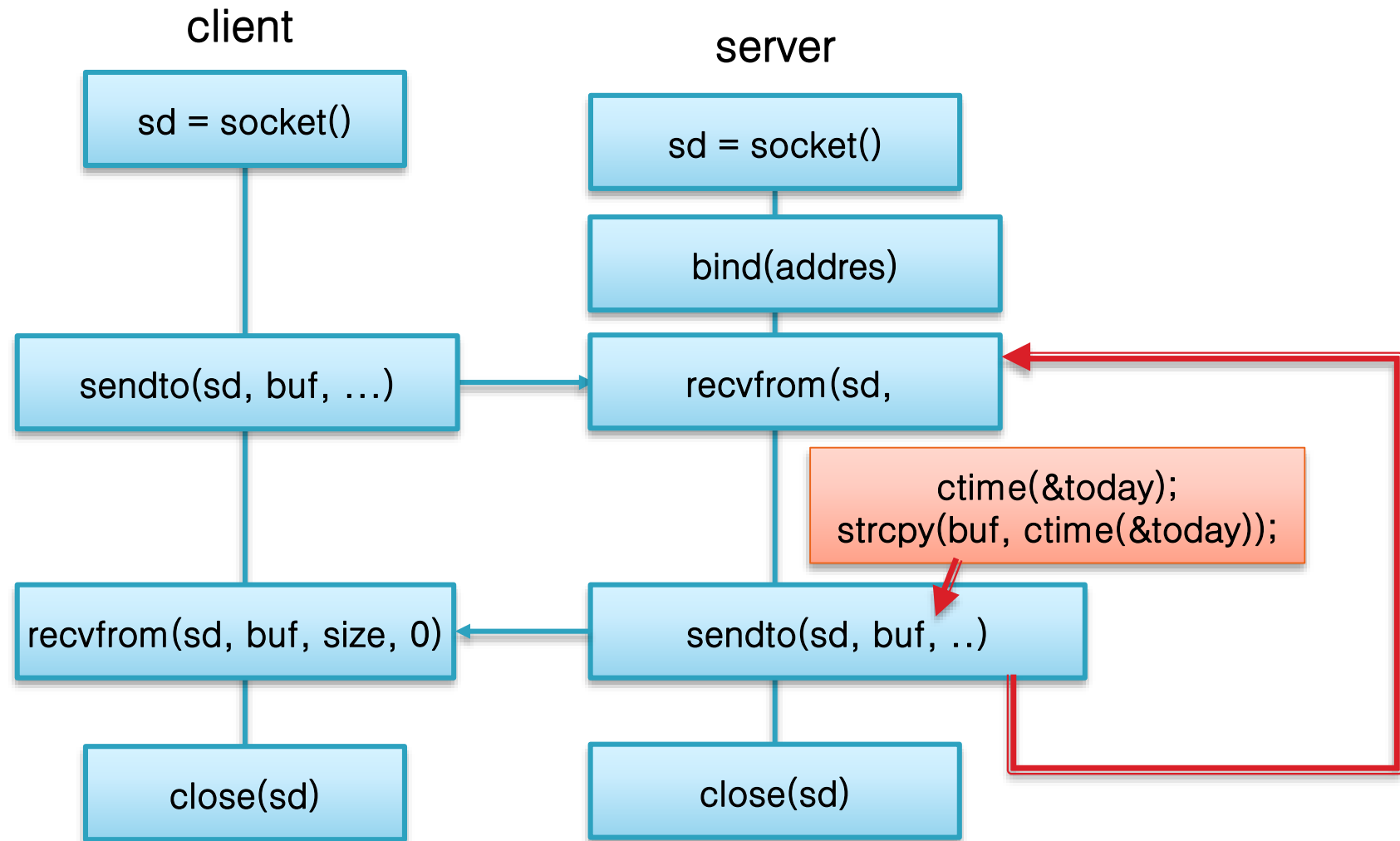
Linux Server

Linux Client

Windows Server

Windows Client

소켓프로그램 예2 - UDP time client/server



소켓프로그램 예2 - UDP time client/server

▶ udp_time_server.c

```
main (int argc, char *argv[])
{
    time_t today;
    char *port_no = TIME_PORT;
    if (argc==2) port_no = argv[1];
    sock = socket (AF_INET, SOCK_DGRAM, 0);
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl (INADDR_ANY);
    server.sin_port = htons (atoi(port_no));
    bind (sock, (struct sockaddr *)&server, sizeof (server));

    while (1) {
        buf_len = recvfrom (sock, buf, 256, 0, (struct sockaddr *)&client, &client_len);
        if (buf_len < 0)
            exit (1);
        printf ("Server: Got %s\n", buf);

        time (&today);
        strcpy (buf, ctime (&today));
        sendto (sock, buf, strlen (buf) + 1, 0, (struct sockaddr *)&client, client_len);
    }
}
```

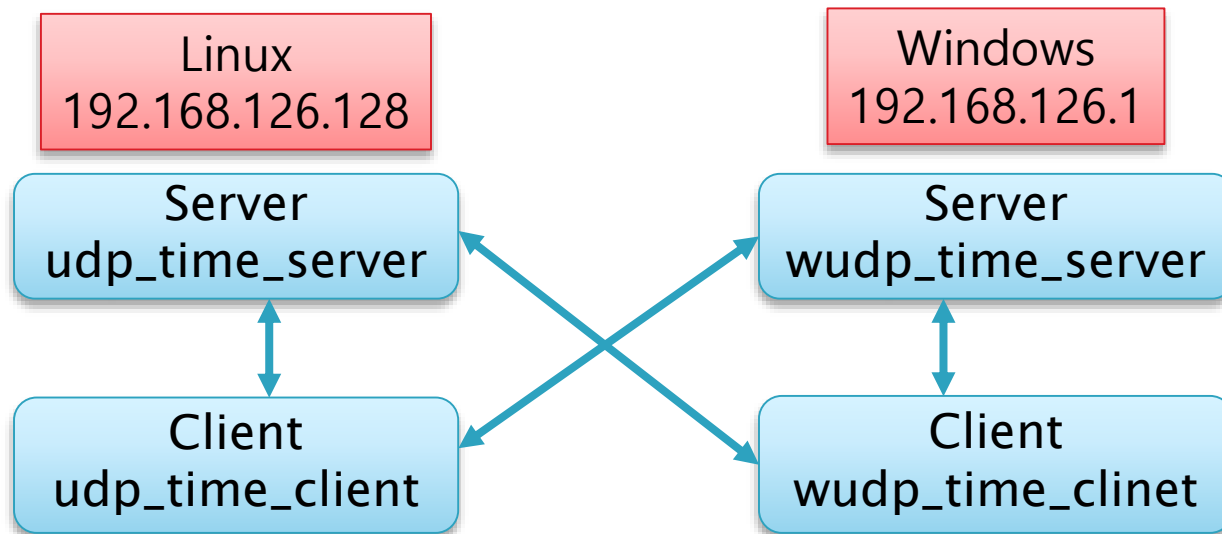
소켓프로그램 예2 - UDP time client/server

▶ udp_time_client.c

```
main ()
{
    sock = socket (AF_INET, SOCK_DGRAM, 0);
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl (inet_addr (TIME_SERVER));
    server.sin_port = htons (TIME_PORT);
    buf[0] = '?'; buf[1] = '\0';
    buf_len = sendto (sock, buf, strlen(buf) + 1, 0, (struct sockaddr *)&server,
        server_len);
    if (buf_len < 0) exit (1);
    buf_len = recvfrom (sock, buf, 256, 0, (struct sockaddr *) 0, (int *) 0);
    if (buf_len < 0) exit (1);
    printf ("Time information from server is %s", buf);
}
```


소켓프로그래밍 예2 - UDP time client/server

- ▶ Linux Client / Linux Server
- ▶ Windows Client / Windows Server
- ▶ Windows Client / Linux Server
- ▶ Linux Client / Windows Server



UDP time client/server 실행 화면

The screenshot displays four terminal windows arranged in a 2x2 grid, illustrating the execution of a UDP time client-server program. Red arrows indicate the flow of data and time synchronization between the components.

- Linux Server (Top Left):** Shows the server process running `./udp_time_server`. It receives two client requests: one from `127.0.0.1` at `18:53:39` and another from `192.168.126.1` at `18:54:55`. The time `18:54:55` is highlighted in pink.
- Linux Client (Bottom Left):** Shows the client process running `./udp_time_client`. It receives time information from the server: `Sat Sep 12 18:53:39 2020` (highlighted in yellow) and `Sat Sep 12 18:54:36 2020`. A large purple 'V' is drawn over the text.
- Windows Server (Bottom Left):** Shows the server process running `cmd.exe`. It receives two client requests: one from `127.0.0.1` at `18:54:08` and another from `192.168.126.128` at `18:54:36`.
- Windows Client (Top Right and Bottom Right):** Shows the client process running `cmd.exe`. It receives time information from the server: `Sat Sep 12 18:54:55 2020` (top right) and `Sat Sep 12 18:54:08 2020` (bottom right).

Red arrows indicate the flow of data and time synchronization between the components:

- From Linux Server to Linux Client (for the first request).
- From Linux Server to Windows Client (for the second request).
- From Windows Server to Windows Client (for the first request).
- From Windows Server to Linux Client (for the second request).

Socket API 정리 (TCP Server)

- ▶ `s = socket (int domain, int type, int protocol)`
 - 매개 변수로 지정된 유형을 지원하는 소켓을 생성
 - 생성된 소켓을 가리키는 파일 디스크립터를 리턴
- ▶ `bind (int s, struct sockaddr *name, socklen_t *namelen)`
 - `s`가 가리키는 소켓에 소켓 주소를 부여함
 - `name`: 소켓 주소 (IP + Port #)
- ▶ `listen (int s, int backlog)`
 - 소켓을 활성화 시킴
- ▶ `accept (int s, struct sockaddr *addr, socklen_t *addrlen)`
 - 클라이언트/서버 환경에서 서버가 대기하는 역할을 함
 - 클라이언트의 `connect()` 함수와 만나면 소켓 연결을 설정함

Socket API 정리 (TCP Client)

- ▶ `s = socket (int domain, int type, int protocol)`
 - 매개 변수로 지정된 유형을 지원하는 소켓을 생성
 - 생성된 소켓을 가리키는 파일 디스크립터를 리턴
- ▶ `connect (int s, struct sockaddr *name, socklen_t namelen)`
 - 클라이언트/서버 환경에서 클라이언트의 연결 설정 요청을 수행함
 - 서버의 `accept()` 함수와 만나면 소켓 연결을 설정함
- ▶ `send (int s, void *msg, size_t len, int flags)`
 - 연결이 설정된 소켓에 데이터를 송신
 - 전송 데이터는 `msg`가 가리킴
- ▶ `recv (int s, void *buf, size_t len, int flags)`
 - 연결이 설정된 소켓에서 데이터를 수신
 - 수신 데이터는 `buf`가 가리키는 공간에 저장됨

개발환경 구축

▶ Linux 개발 환경 구축

- 가상머신 이용
 - Vmware + Linux(Ubuntu, CentOS, ..)
 - VirtualBox + Linux
- Linux 전용 PC(Notebook) 사용
 - 이 경우 2대의 PC 가 필요함 (Linux 전용, Windows 전용)

▶ Macbook 의 경우

- Unix 기반 (Linux 99.9% 호환)
- Windows PC가 별도로 있다면 (Unix / Windows 통신)
- PC가 없다면 Unix to Unix Local 통신으로 실습
- C 언어 사용 가능하면 됨

Vmware + Ubuntu Linux 설치

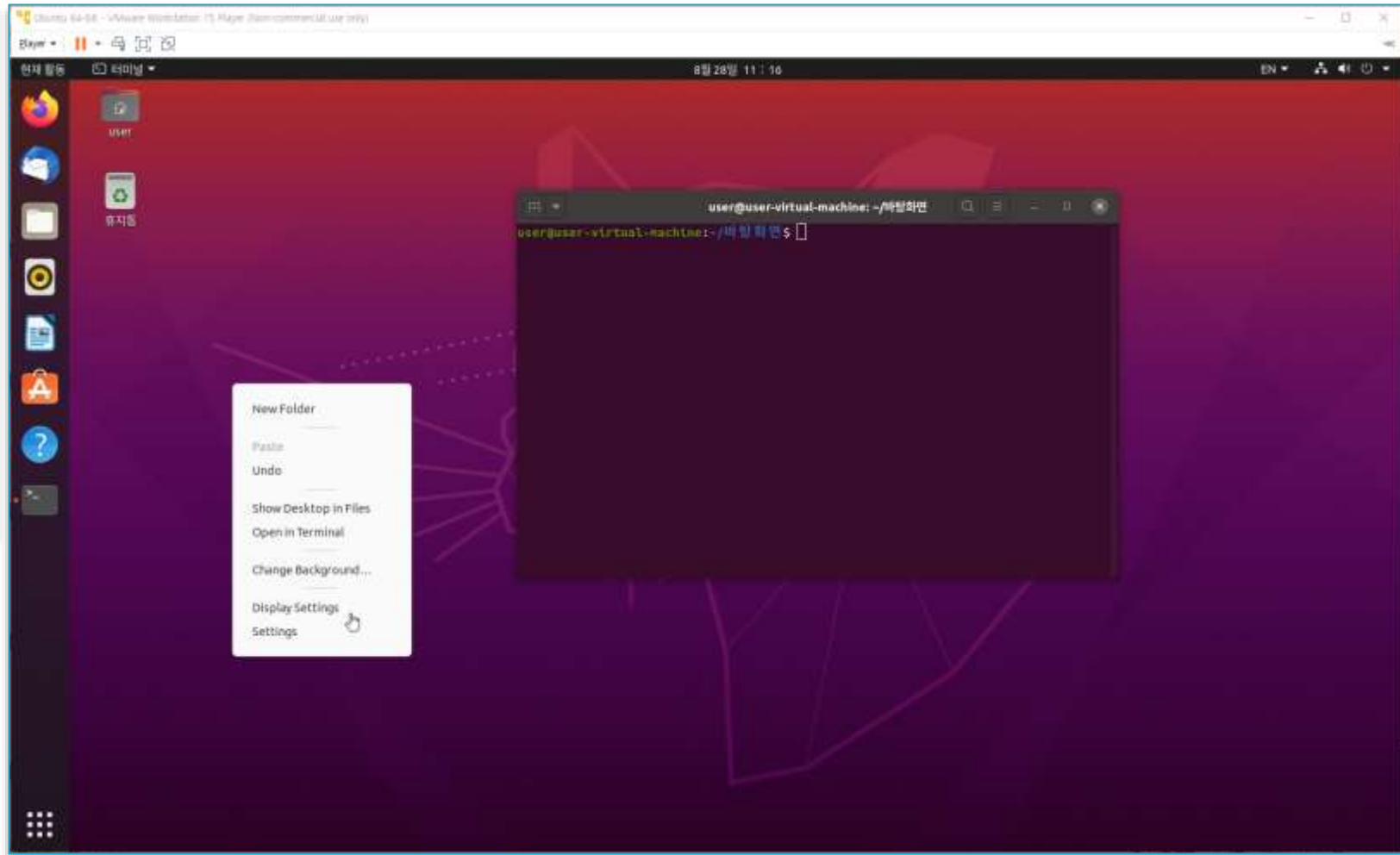
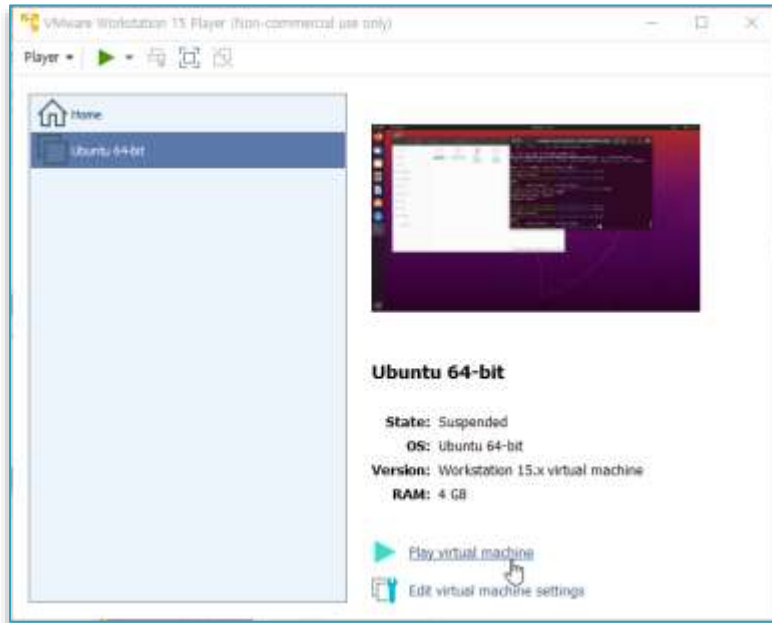
▶ Ubuntu 다운

- <https://ubuntu.com/#download>
- ubuntu-20.04-desktop-amd64.iso

▶ Vmware 다운 설치, 실행

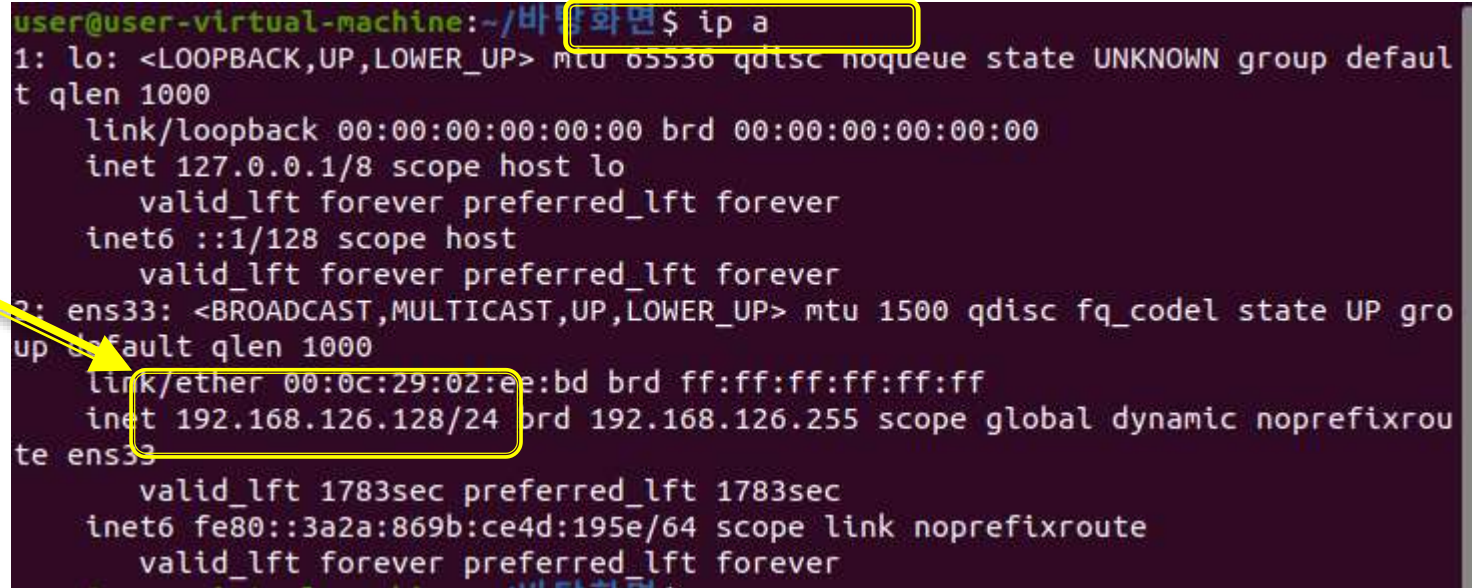
- <https://www.vmware.com/kr/products/workstation-player/workstation-player-evaluation.html>
- VMware-player-15.5.6-16341506.exe
- VMware 실행
 - Create New Virtual Machine
 - ubuntu-20.04-desktop-amd64.iso 설정
 - Ubuntu Virtual Machine 실행
 - Ubuntu 설치 진행

Vmware Linux 실행



Ubuntu 설정

- ▶ gcc, make
 - sudo apt install gcc
 - sudo apt install make
- ▶ Network tools (ifconfig, netstat ..) 설치
 - sudo apt install net-tools
- ▶ Network 환경 확인
 - ip a 로 확인
 - 192.168.126.128



```
user@user-virtual-machine:~/바$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:02:ee:bd brd ff:ff:ff:ff:ff:ff
    inet 192.168.126.128/24 brd 192.168.126.255 scope global dynamic noprefixroute ens33
        valid_lft 1783sec preferred_lft 1783sec
    inet6 fe80::3a2a:869b:ce4d:195e/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

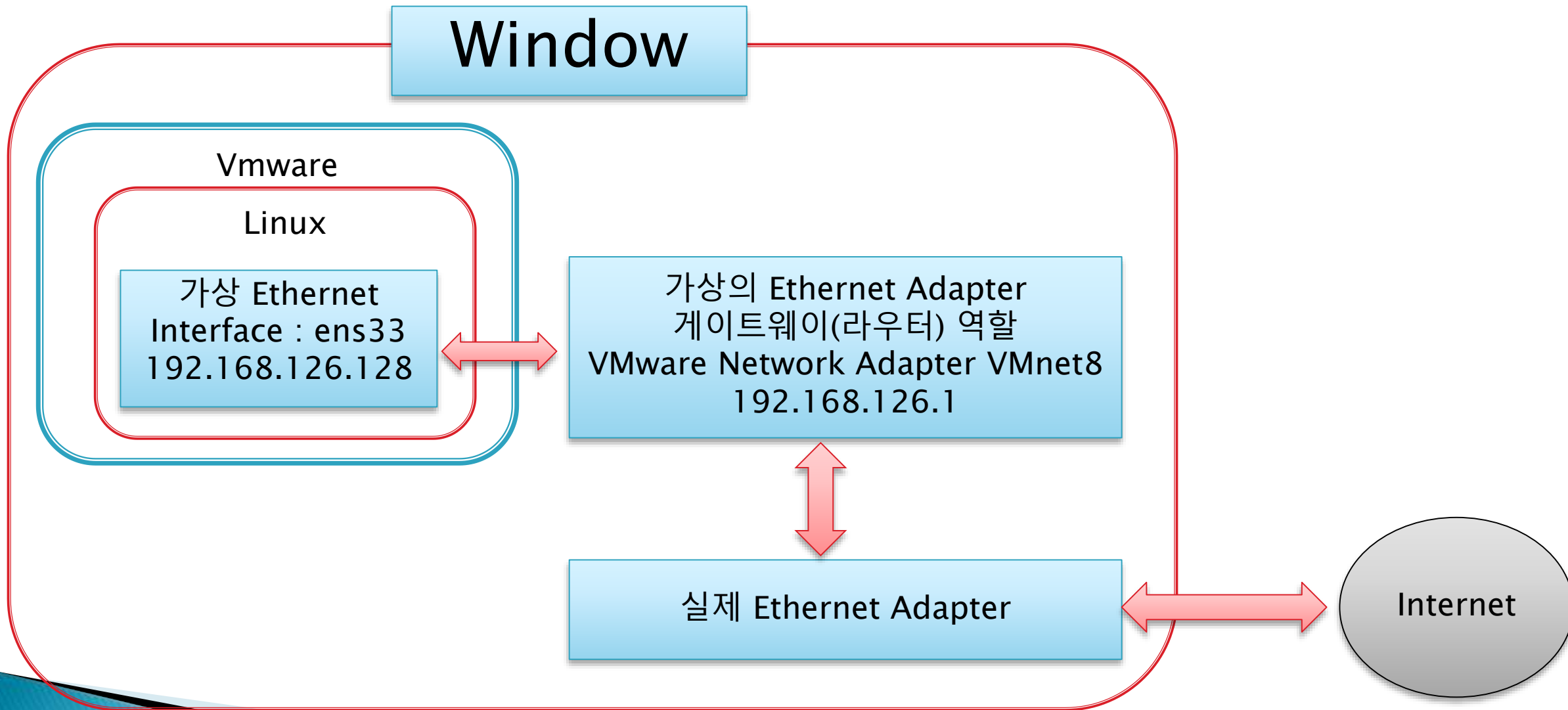

ifconfig 로 IP 확인

```
user@user-virtual-machine: ~/바탕화면
user@user-virtual-machine:~/바탕화면$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.126.128 netmask 255.255.255.0 broadcast 192.168.126.255
    inet6 fe80::2a2a:869b:ce4d:195e prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:02:ee:bd txqueuelen 1000 (Ethernet)
    RX packets 122318 bytes 162141160 (162.1 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 60690 bytes 4222246 (4.2 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 1754 bytes 169230 (169.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1754 bytes 169230 (169.2 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

user@user-virtual-machine:~/바탕화면$
```

Linux / Windows Network 원리



Vmware 설치되면 Virtual Ethernet Adapter 생성됨

네트워크 연결

제어판 > 네트워크 및 인터넷 > 네트워크 연결

구성 이 네트워크 장치 사용 이 연결 진단 이 연결 이름 바꾸기 이 연결의 설정 변경

이름	상태	장치 이름
VMware Network Adapter VMnet1	사용함	VMware Virtual Ethernet Adapter for VMnet1
VMware Network Adapter VMnet8	사용함	VMware Virtual Ethernet Adapter for VMnet8
Wi-Fi	KT_GiGA_5G_Wave2_4531	Realtek 8812AU Wireless LAN 802.11ac USB NIC
이더넷	사용 안 함	Realtek PCIe GbE Family Controller

VMnet1

외부로 나갈 수 없는 내부용

192.168.237.1

VMnet8

외부로 나갈 수 있는 게이트웨이(라우터) 역할

192.168.126.1

4개 항목 1개 항목 선택함

명령 프롬프트

```
C:\Users\Daddy>ipconfig
```

Windows IP 구성

무선 LAN 어댑터 Wi-Fi:

미디어 상태 : 미디어 연결 끊김
연결별 DNS 접미사 :

무선 LAN 어댑터 로컬 영역 연결* 2:

미디어 상태 : 미디어 연결 끊김
연결별 DNS 접미사 :

무선 LAN 어댑터 로컬 영역 연결* 12:

미디어 상태 : 미디어 연결 끊김
연결별 DNS 접미사 :

이더넷 어댑터 이더넷:

연결별 DNS 접미사 :
링크-로컬 IPv6 주소 : fe80::95c2:8db1:aa34:1b54%15
IPv4 주소 : 172.30.1.33
서브넷 마스크 : 255.255.255.0
기본 게이트웨이 : 172.30.1.254

이더넷 어댑터 VMware Network Adapter VMnet1:

연결별 DNS 접미사 :
링크-로컬 IPv6 주소 : fe80::d1a8:7a11:815d:4d2f%21
IPv4 주소 : 192.168.237.1
서브넷 마스크 : 255.255.255.0
기본 게이트웨이 :

이더넷 어댑터 VMware Network Adapter VMnet8:

연결별 DNS 접미사 :
링크-로컬 IPv6 주소 : fe80::6463:3a4b:5f95:e458%8
IPv4 주소 : 192.168.126.1
서브넷 마스크 : 255.255.255.0
기본 게이트웨이 :

3주 과제

- ▶ Linux 개발환경 구축
 - Vmware 설치
 - Ubuntu Linux 설치 및 환경 설정
 - Ubuntu Linux 개발 환경 프로그램 설치
 - make, gcc, net-tools
- ▶ time client/server 실습 화면 capture
 - TCP Client/Server
 - UDP Client/Server
 - Linux / Windows 간 실습 (총 8가지 TCPx4, UDPx4)
 - Linux Client/Linux Server
 - Windows Client/Linux Server
 - Windows Client/Windows Server
 - Linux Client/Windows Server
- ▶ Linux 또는 Windows 가 안되는 경우
 - Linux / Linux 또는
 - Windows / Windows 로 실습