인하공업전문대학
INHA TECHNICAL COLLEGE

TCP/IP
네트워크 프로그래밍
6주차

인하공업전문대학 컴퓨터 정보과

김한결 강사

목차

- TCP기반 서버, 클라이언트 구현
- 네트워크 데이터 송수신 주체
- 프로토콜
- TCP/IP 프로토콜
- 클라우드 서비스

기본 TCP 서버 프로그램

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

void error_handling(char *message);
```

```c
int main(int argc, char *argv[])
{
    int serv_sock;
    int clnt_sock;

    struct sockaddr_in serv_addr;
    struct sockaddr_in clnt_addr;
    socklen_t clnt_addr_size;

    char message[] = "Hello World!";

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

    serv_sock = socket(PF_INET, SOCK_STREAM, 0);
    if(serv_sock == -1)
        error_handling("socket() error");
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_addr.sin_port = htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr))==-1)
        error_handling("bind() error");
    if(listen(serv_sock,5)==-1)
        error_handling("listen() error");

    clnt_addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_addr, &clnt_addr_size)
;
    if(clnt_sock==-1)
        error_handling("accept() error");
    write(clnt_sock, message, sizeof(message));
    close(clnt_sock);
    close(serv_sock);
    return 0;
}

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

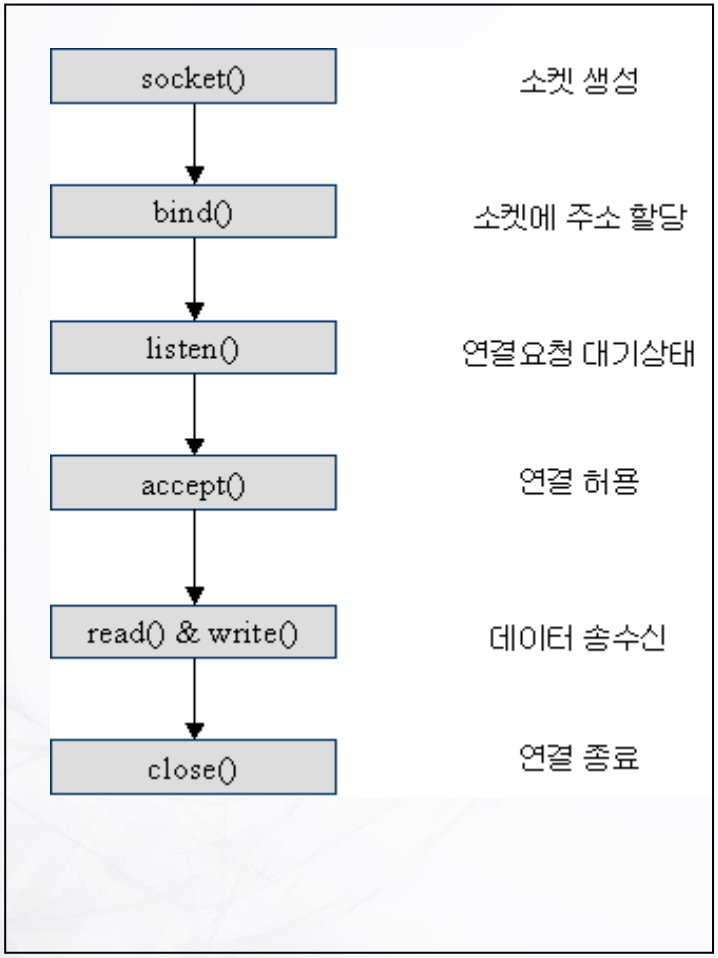
47,1 87%

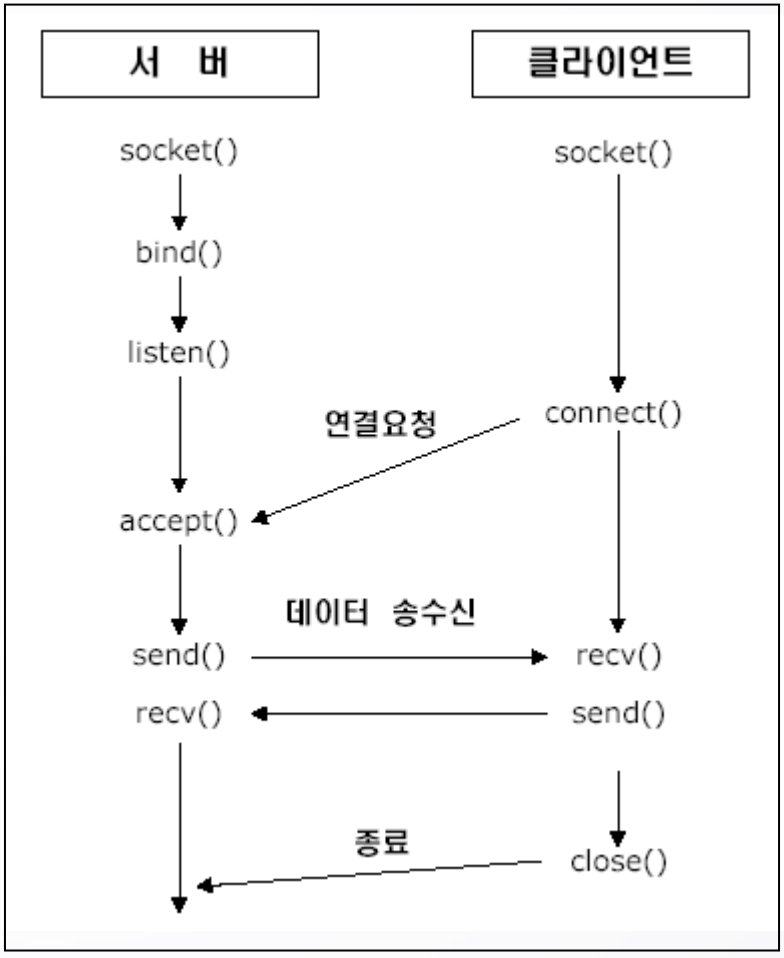
기본 TCP 클라이언트 프로그램

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

void error_handling(char *message);

int main(int argc, char *argv[])
{
    int sock;

    struct sockaddr_in serv_addr;
    char message[30];
    int str_len;

    if(argc!=3)
    {
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }

    sock = socket(PF_INET, SOCK_STREAM, 0);
    if(sock == -1)
        error_handling("socket() error");
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));

    if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
            error_handling("socket() error");
    str_len=read(sock,message,sizeof(message)-1);
    if(str_len==-1)
            error_handling("read() error!");
    printf("Message from server : %s \n", message);
    close(sock);
    return 0;
}

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

TCP기반 서버, 클라이언트 구현

- TCP 서버에서의 기본적인 함수 호출 순서





- 소켓생성

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

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

domain : 소켓이 사용할 프로토콜 체계(Protocol Family) 정보 전달.

type : 소켓의 데이터 전송방식에 대한 정보 전달.

protocol : 두 컴퓨터간 통신에 사용되는 프로토콜 정보 전달.

-> 하나의 프로토콜 체계 안에 데이터의 전송방식이 동일한 프로토콜 둘 이상 존재.(IPPRO_TCP , IPPRO_UDP)

이름	프로토콜 체계
PF_INET	IPV4 인터넷 프로토콜 체계
PF_INET6	IPV6 인터넷 프로토콜 체계
PF_LOCAL	로컬통신을 위한 UNIX 프로토콜 체계
PF_PACKET	Low Level 소켓을 위한 프로토콜 체계
PF_IPX	IPX 노벨 프로토콜 체계

- **int type**

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

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

- **연결지향형 소켓(SOCK_STREAM) 특징**
  I. 중간에 데이터가 소멸되지 않고 목적지로 전송
  II. 전송 순서대로 데이터가 수신
  III. 전송되는 데이터의 경계(Boundary)가 존재하지 않음.
  IV. 소켓 대 소켓의 연결은 반드시 1대 1이어야함.

-> 신뢰성 있는 순차적인 바이트 기반의 연결지향 데이터 전송 방식의 소켓

- **int type**

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

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

- **연결지향형 소켓(SOCK_STREAM) 특징**
  I. 중간에 데이터가 소멸되지 않고 목적지로 전송
  II. 전송 순서대로 데이터가 수신
  III. 전송되는 데이터의 경계(Boundary)가 존재하지 않음.
  IV. 소켓 대 소켓의 연결은 반드시 1대 1이어야함.

-> 신뢰성 있는 순차적인 바이트 기반의 연결지향 데이터 전송 방식의 소켓

- **int type**

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

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

- **비연결지향형 소켓(SOCK_DGRAM) 특징**
  I. 전송된 데이터는 손실의 우려가 있고, 파손의 우려가 있음
  II. 전송되는 데이터의 경계(Boundary)가 존재한다.
  III. 한번에 전송할 수 있는 데이터의 크기가 제한된다.

-> 신뢰성과 순차적 데이터 전송을 보장하지 않는,
고속 데이터 전송을 목적으로 하는 소켓

- **SOCK_STREAM vs SOCK_DGRAM**

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

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

SOCK_STREAM	SOCK_DGRAM
중간에 데이터 소멸되지 않고, 목적지로 전송	전송된 데이터는 손실의 우려가 있고, 파손의 우려가 있음
데이터의 경계가 존재하지 않음	데이터의 경계가 존재함
UDP에 비해, 전송속도가 느림	TCP에 비해 전송속도가 빠름
한번에 전송되는 데이터 크기에 제한이 없음	한번에 전송되는 데이터 크기에 제한이 있음
버퍼가 존재함	버퍼가 존재하지 않음

TCP기반 서버, 클라이언트 구현

- 소켓생성

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

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

  - 성공 시 파일디스크립터, 실패 시 -1 반환

  - int serv_sock;
  - serv_sock = socket(PF_INET, SOCK_STREAM, 0);
  - If(serv_sock == -1)
    error_handling("socket() error");

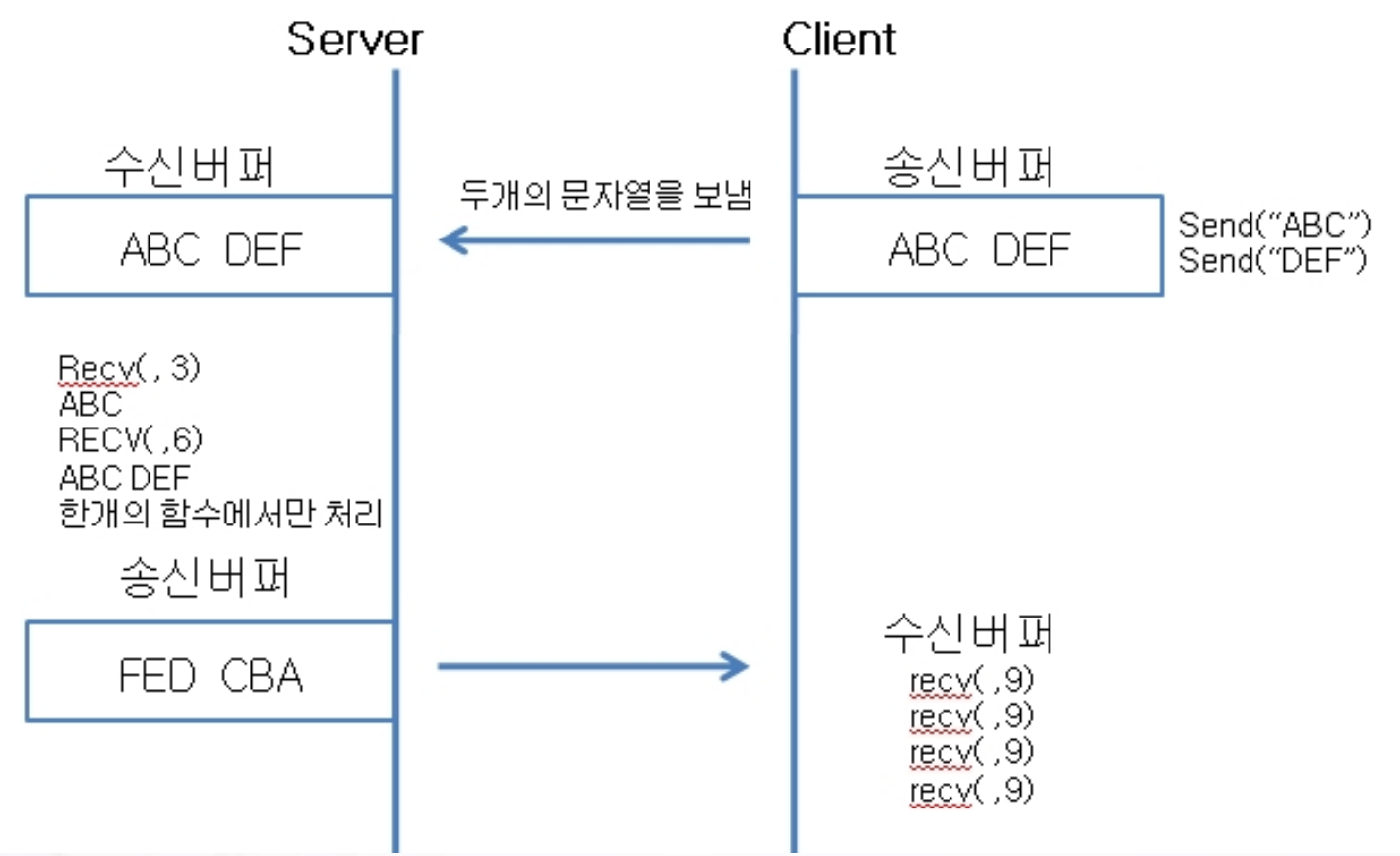
TCP 클라이언트 프로그램 (데이터의 경계가 존재하지 않음)

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

void error_handling(char *message);

int main(int argc, char* argv[])
{
    int sock;
    struct sockaddr_in serv_addr;
    char message[30];
    int str_len=0;
    int idx=0, read_len=0;

    if(argc!=3){
        printf("Usage : %s <IP> <port>\n", argv[0]);
        exit(1);
    }

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

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

    if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
        error_handling("connect() error!");

```

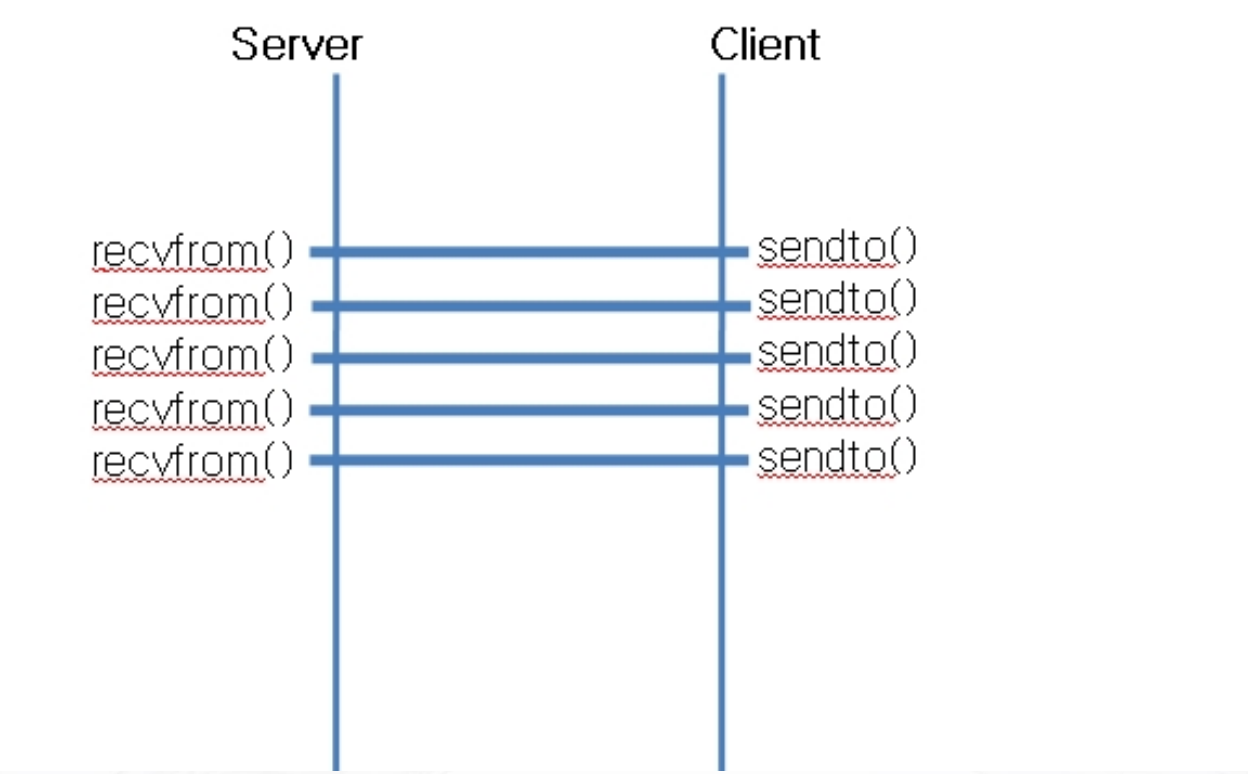
TCP 클라이언트 프로그램 (데이터의 경계가 존재하지 않음)

```
36      while(read_len=read(sock, &message[idx++], 1))
37      {
38          if(read_len==-1)
39              error_handling("read() error!");
40
41          str_len+=read_len;
42      }
43
44      printf("Message from server: %s ₩n", message);
45      printf("Function read call count: %d ₩n", str_len);
46      close(sock);
47      return 0;
48  }
49
50  void error_handling(char *message)
51  {
52      fputs(message, stderr);
53      fputc('₩n', stderr);
54      exit(1);
55  }
                                                    55,1          Bot
```

TCP


Server
Client
수신버퍼
ABC DEF
두개의 문자열을 보냄
송신버퍼
ABC DEF
Send("ABC")
Send("DEF")
Recv(, 3)
ABC
RECV(,6)
ABC DEF
한개의 함수에서만 처리
송신버퍼
FED CBA
수신버퍼
recv(,9)
recv(,9)
recv(,9)
recv(,9)

UDP


Server
Client
recvfrom()
recvfrom()
recvfrom()
recvfrom()
recvfrom()
sendto()
sendto()
sendto()
sendto()
sendto()

리눅스 파일 디스크립터

- 리눅스에서의 소켓조작은 파일조작과 동일
- 소켓을 파일의 일종으로 구분
- 리눅스에서 제공

- 표준 파일 디스크립터는 별도의 생성과정을 거치지 않아도 프로그램을 실행하면 자동으로 할당되는 파일 디스크립터

파일 디스크립터	대 상
0	표준입력: Standard Input
1	표준출력 : Standard Output
2	표준에러 : Standard Error

- 파일 열기

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

int open(const char *path, int flag);
```

path : 파일 이름을 나타내는 문자열의 주소 값 전달.

flag : 파일의 오픈 모드 정보 전달.

오픈 모드	의 미
O_CREAT	필요하면 파일을 생성
O_TRUNC	기존 데이터 전부 삭제
O_APPEND	기존데이터보존, 뒤에 추가 하여 저장
O_RDONLY	읽기전용으로 파일 오픈
O_WRONLY	쓰기전용으로 파일 오픈
O_RDWR	읽기,쓰기 겸용으로 파일 오픈

- 파일 닫기

```
#include <unistd.h>

int close(int fd);
```

fd : 닫고자 하는 파일 또는 소켓의 파일 디스크립터 전달.

- 파일에 데이터 쓰기

```
#include <unistd.h>

ssize_t write(int fd, const void * buf, size_t nbytes);
```

fd : 데이터 전송대상을 나타내는 파일 디스크립터 전달.

buf : 전송할 데이터가 저장된 버퍼의 주소 값 전달

nbytes : 전송할 데이터의 바이트 수 전달

파일 디스크립터 생성 예제 - 1

- low_open.c

~/socket_programing

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <fcntl.h>
 4 #include <unistd.h>
 5
 6 void error_handling(char * message);
 7
 8 int main(void)
 9 {
10     int fd;
11     char buf[]="Let's go!\n";
12     fd=open("data.txt", O_CREAT|O_WRONLY|O_TRUNC);
13     if(fd==-1)
14         error_handling("open() error!");
15     printf("File descripter : %d \n", fd);
16
17     if(write(fd, buf, sizeof(buf))==-1)
18         error_handling("Write() error!");
19     close(fd);
20     return 0;
21 }
22
23 void error_handling(char *message)
24 {
25     fputs(message, stderr);
26     fputc('\n', stderr);
27     exit(1);
28 }
```

12,1-4 꼭 대 기

파일 디스크립터 생성 예제 - 2

- low_read.c

~/socket_programing

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#define BUF_SIZE 100

void error_handling(char * message);

int main(void)
{
    int fd;
    char buf[BUF_SIZE];


    fd=open("data.txt", O_RDONLY);
    if(fd==-1)
        error_handling("open() error!");
    printf("File descripter : %d \n", fd);

    if(read(fd, buf, sizeof(buf))==-1)
        error_handling("read() error!");
    printf("file data: %s", buf);
    close(fd);
    return 0;
}

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

```

파일 디스크립터 생성 예제 - 3

- fd_seri.c

~/socket_programing

```
 1 #include <stdio.h>
 2 #include <fcntl.h>
 3 #include <unistd.h>
 4 #include <sys/socket.h>
 5
 6 int main(void)
 7 {
 8     int fd1,fd2,fd3;
 9     fd1 = socket(PF_INET, SOCK_STREAM, 0);
10     fd2 = open("test.dat",O_CREAT|O_WRONLY|O_TRUNC);
11     fd3 = socket(PF_INET, SOCK_DGRAM, 0);
12
13     printf("file descripter 1: %d\n", fd1);
14     printf("file descripter 2: %d\n", fd2);
15     printf("file descripter 3: %d\n", fd3);
16
17     close(fd1);
18     close(fd2);
19     close(fd3);
20
21     return 0;
22
23 }
                                                    4,1           모 두
```

- 연결 요청 대기 상태로의 진입

 # listen 함수는 전달되는 인자의 소켓을 '서버 소켓'이 되게 한다.

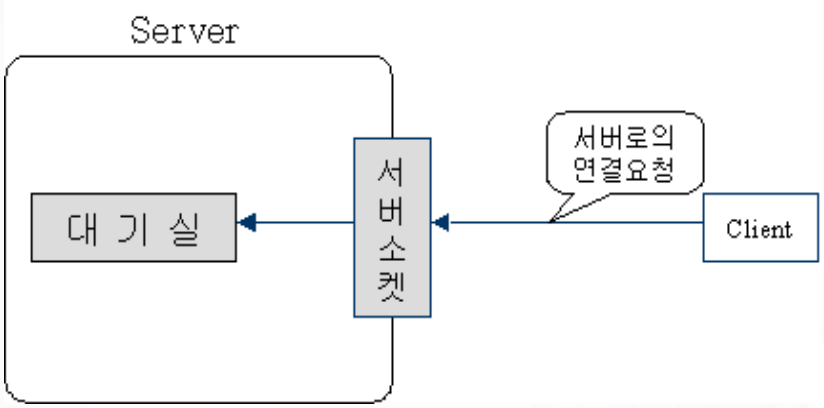
 # listen 함수는 '연결 요청 대기 큐'를 생성 한다.

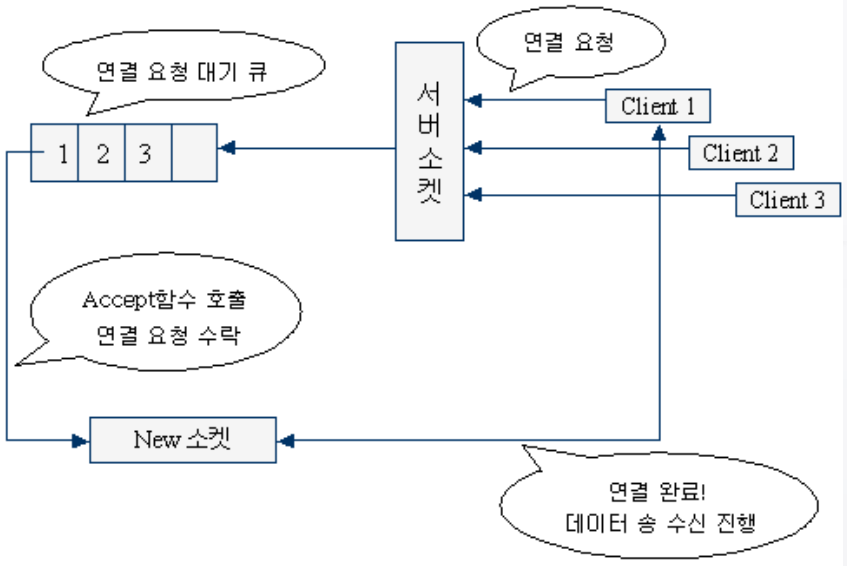
```
#include <sys/type.h>

int listen(int sock, int backlog);
```

sock : 연결요청 대기상태에 두고자 하는 소켓의 파일 디스크립터,
디스크립터 소켓의 서버 소켓 (리스닝 소켓)이 된다.
backlog: 연결요청 대기 큐의 크기 정보
5가 되면 큐의 크기가 5가 되어 클라이언트의 요청을 5개까지
대기시킬 수 있다.

- 서버의 역할과 연결요청 대기상태

 # 서버 소켓은 일종의 '문지기' 이다.





- 연결요청 수락하기

 # 연결요청 대기 큐(queue)에 존재하는 클라이언트의 연결 요청 수락.

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

int accept(int sock, struct sockaddr * addr, int * addrlen);
```

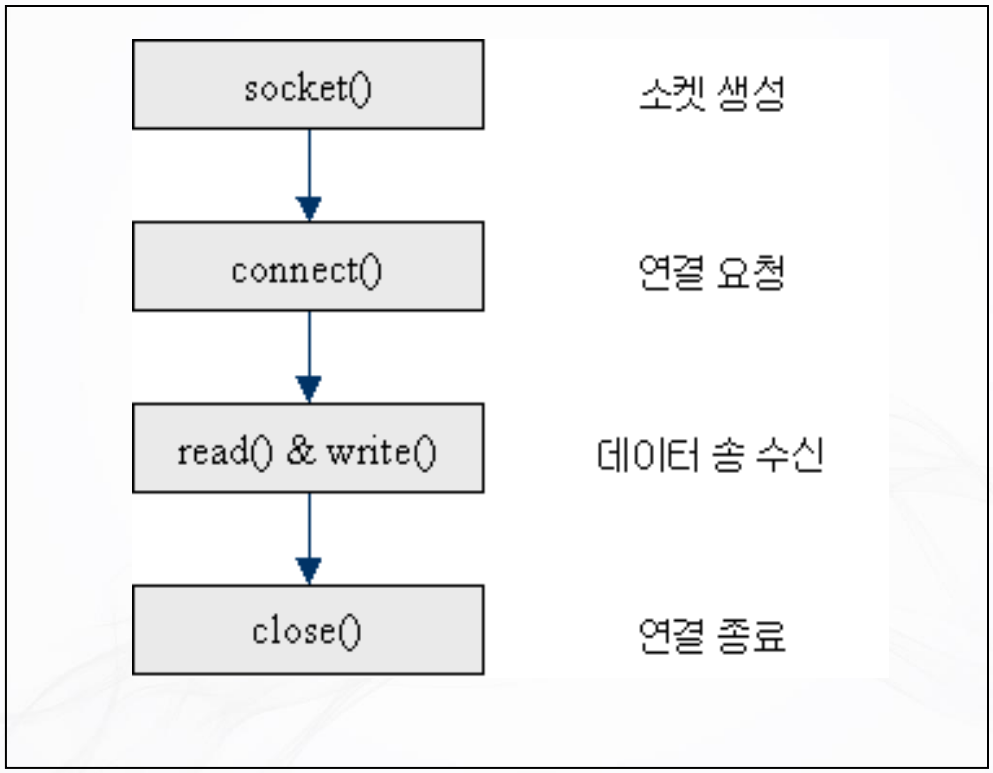
sock : 서버 소켓의 파일 디스크립터,

addr: **연결 요청한 클라이언트의 주소 정보를 담을 변수의 주소 값 전달**
함수 호출이 완료되면 인자로 전달된 주소의 변수에는 클라이언트의 주소 정보가 채워짐

addrlen: addr에 전달된 주소의 변수 크기를 바이트 단위로 전달

TCP기반 서버, 클라이언트 구현

- 클라이언트의 기본적인 함수 호출 순서



- 연결 요청 함수

 # 소켓과 목적지 주소에 대한 정보를 마련 해 두고 나서 연결 요청을 시도 한다.

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

int connect(int sock, struct sockaddr *servaddr, socklen_t addrlen);
```

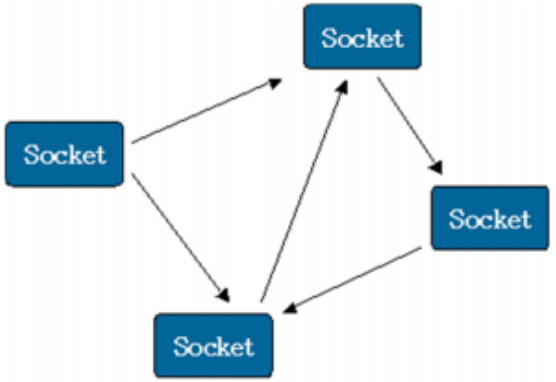
sock : 클라이언트 소켓의 파일 디스크립터,

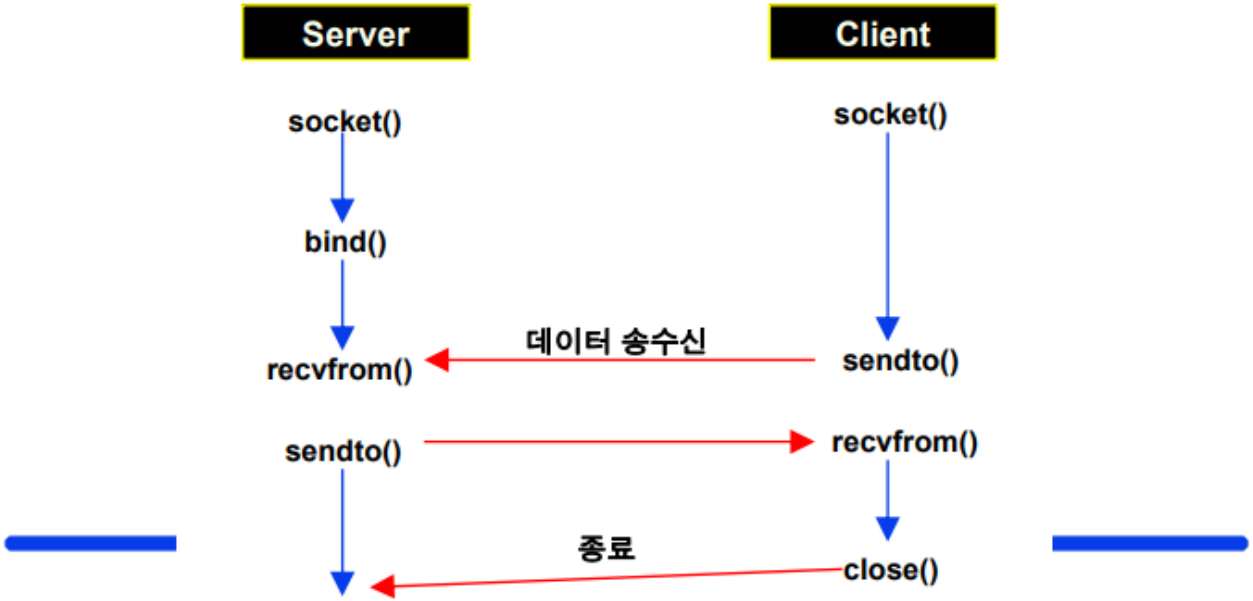
servaddr: 연결 요청할 서버의 주소 정보를 담은 변수의 주소 값

addrlen: servaddr에 전달된 주소의 변수 크기를 바이트 단위로 전달

UDP 소켓프로그래밍

- TCP와 달리 일 대 일 통신에만 사용되지 않음
- 비연결형 소켓
  - 일반적으로 연결 설정 과정을 거치지 않음
  - 데이터를 위한 소켓은 하나만 개설
  - 소켓 개설 후 바로 상대방과 데이터를 송수신





UDP 기반의 데이터 입출력 함수

- 데이터 전송 함수 → 주소 정보를 포함해서 데이터를 전송하는 함수

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

  - sock: 데이터를 전송할 때 사용할 소켓의 파일 디스크립터
  - msg: 전송하고자 하는 데이터를 저장해 놓은 버퍼를 가리키는 포인터
  - len: msg 포인터가 가리키는 위치에서부터 몇 바이트를 전송할 것인지의 크기
  - flags: 옵션을 설정하는데 필요한 인자. 일반적으로 0.
  - addr: 전송하고자 하는 곳의 주소 정보
  - addrlen: addr 포인터가 가리키고 있는 구조체 변수의 크기

UDP 기반의 데이터 입출력 함수

- 데이터 수신 함수 → 데이터가 전송된 위치 정보를 얻을 수 있는 기능 제공

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

  - sock: 데이터를 수신할 때 사용할 소켓의 파일 디스크립터
  - buf: 수신할 데이터를 저장할 버퍼를 가리키는 포인터
  - len: 수신할 수 있는 최대 바이트 수
  - flags: 옵션을 설정하는데 필요한 인자. 일반적으로 0.
  - addr: 주소 정보 구조체 변수의 포인터. 함수 호출이 끝나면, 데이터를 전송한 호스트의 주소 정보로 채워짐
  - addrlen: addr 포인터가 가리키고 있는 구조체 변수의 크기

UDP 통신 (udp_server.c)

서버프로그램

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

#define BUFSIZE 30

void error_handling(char *message);

int main(int argc, char *argv[])
{
        int serv_sock;
        char message[BUFSIZE];
        int str_len,num =0;
        struct sockaddr_in serv_addr;
        struct sockaddr_in clnt_addr;
        int clnt_addr_size;

        serv_sock = socket(PF_INET, SOCK_DGRAM,0);

        if(serv_sock == -1)
        {
           error_handling("socket() error");
        }
        memset(&serv_addr, 0, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
        serv_addr.sin_port = htons(atoi(argv[1]));

        if(bind(serv_sock,(struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1)
        {
                error_handling("bind() error");
        }
        sleep(1);

        while(1)
        {
                clnt_addr_size = sizeof(clnt_addr);
                sleep(1);
                str_len = recvfrom(serv_sock, message, BUFSIZE, 0, (struct sockaddr*)&clnt_addr,&clnt_addr_size);
                printf("수신번호 : %d₩n", num++);
                sendto(serv_sock, message,str_len,0,(struct sockaddr *) &clnt_addr, sizeof(clnt_addr));

        }
        return 0;
}
void error_handling(char *message)
{
        fputs(message, stderr);
        fputc('₩n', stderr);
        exit(1);
}
```

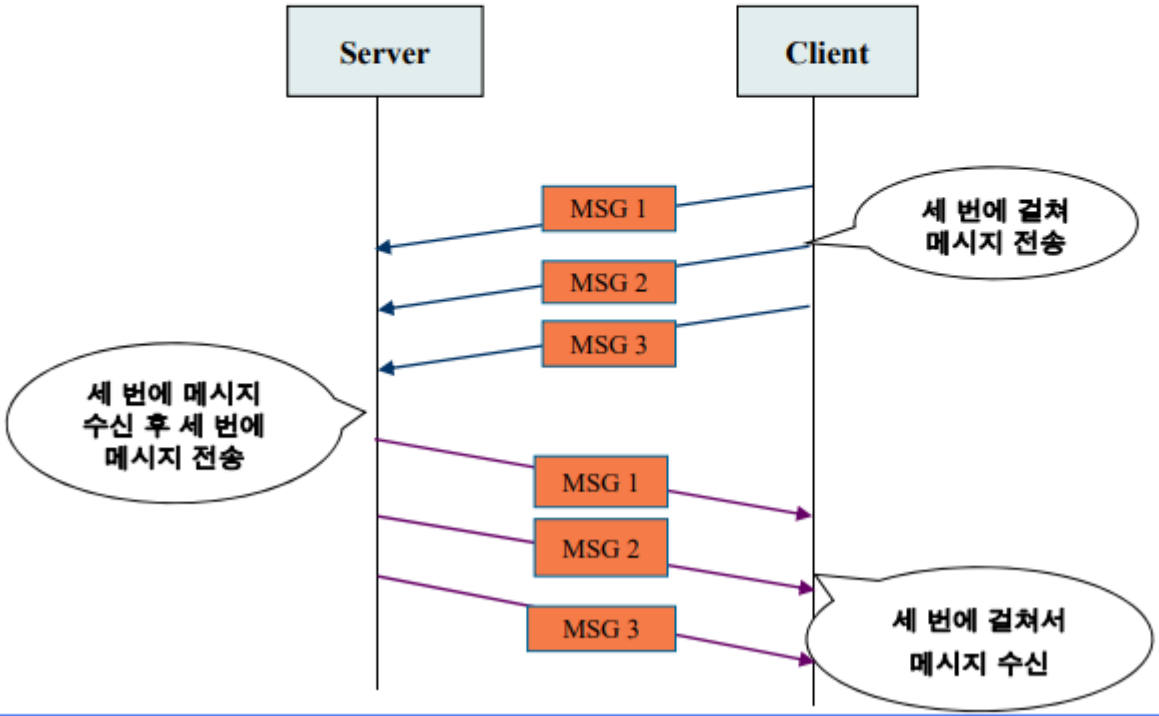
UDP 통신 (udp_client.c)

클라이언트 프로그램

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

#define BUFSIZE 30

void error_handling(char *message);

int main(int argc, char *argv[])
{
        int clint_sock;
        char message[BUFSIZE];
        int str_len,addr_size,i;
        char msg1[] = "good";
        char msg2[] = "afternoon!";
        char msg3[] = "everybody!";
        struct sockaddr_in serv_addr;
        struct sockaddr_in from_addr;
        int clnt_addr_size;

        clint_sock = socket(PF_INET, SOCK_DGRAM,0);

        if(clint_sock == -1)
        {
            error_handling("socket() error");
        }
        memset(&serv_addr, 0, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
        serv_addr.sin_port = htons(atoi(argv[2]));

        sendto(clint_sock,msg1,strlen(msg1),0,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
        sendto(clint_sock,msg2,strlen(msg2),0,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
        sendto(clint_sock,msg3,strlen(msg3),0,(struct sockaddr*)&serv_addr,sizeof(serv_addr));

        for(i=0; i<3; i++)
        {
                addr_size = sizeof(from_addr);
                str_len = recvfrom(clint_sock,message,BUFSIZE,0,(struct sockaddr*)&from_addr, &addr_size);
                message[str_len] = 0;
                printf("서버로부터 전송된 메세지 : %s ₩n",message);
        }
        close(clint_sock);

        return 0;
}
void error_handling(char *message)
{
        fputs(message, stderr);
        fputc('₩n', stderr);
        exit(1);
}
```

UDP 소켓 (데이터 경계가 존재함)

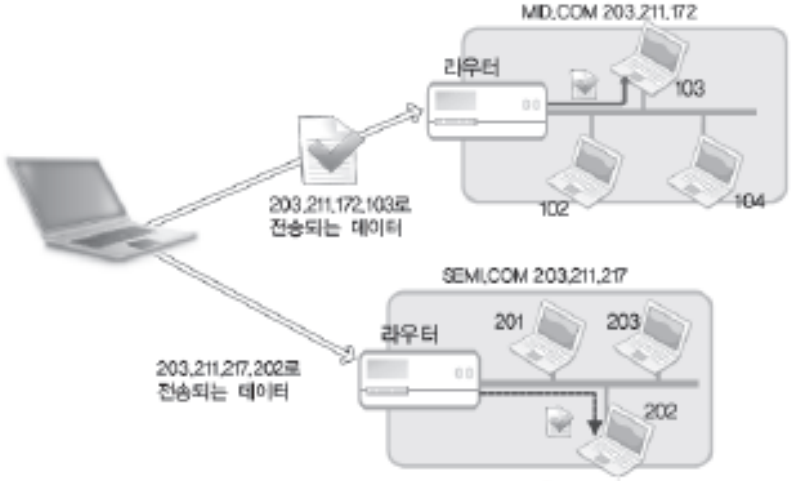
- 한쪽이 sendto()를 호출했으면 이 데이터를 받기 위해서 상대방은 반드시 recvfrom()을 호출해야 함
- TCP 소켓에서는 스트림을 이용하므로 write()나 send()로 연속하여 쓰기를 수행하거나 read()나 recv()로 연속하여 읽기를 해도 문제가 되지 않음
- UDP에서는 sendto() 와 recvfrom() 호출이 서로 짝을 이루도록 순서가 맞아야 함



인터넷 주소(Internet Address)

- 인터넷 주소란?
  - 인터넷상에서 컴퓨터를 구분하는 목적으로 사용되는 주소.
  - 4바이트 주소체계인 IPv4와 16바이트 주소체계인 IPv6가 존재한다.
  - 소켓을 생성할 때 기본적인 프로토콜을 지정해야 한다.
  - 네트워크 주소와 호스트 주소로 나뉜다. 네트워크 주소를 이용해서 네트워크를 찾고, 호스트 주소를 이용해서 호스트를 구분한다.

IPv4 인터넷 주소의 체계



인터넷 주소의 역할

클래스 별 네트워크 주소와 호스트 주소의 경계

- 클래스 A의 첫 번째 바이트 범위 0이상 127이하
- 클래스 B의 첫 번째 바이트 범위 128이상 191이하
- 클래스 C의 첫 번째 바이트 범위 192이상 223이하

달리 말하면...

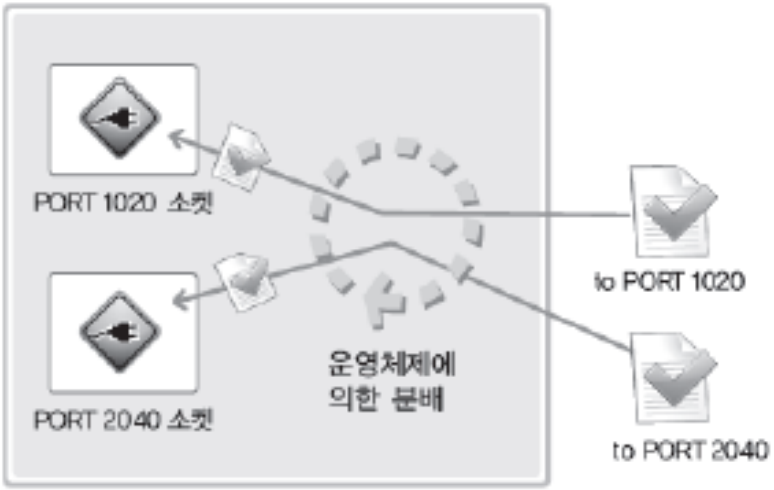
- 클래스 A의 첫 번째 비트는 항상 0으로 시작
- 클래스 B의 첫 두 비트는 항상 10으로 시작
- 클래스 C의 첫 세 비트는 항상 110으로 시작

때문에 첫 번째 바이트 정보만 참조해도 IP주소의 클래스 구분이 가능하며,

이로 인해서 네트워크 주소와 호스트 주소의 경계 구분이 가능하다.

소켓의 구분에 활용되는 PORT번호

- PORT번호
  - IP는 컴퓨터를 구분하는 용도로 사용되며, PORT번호는 소켓을 구분하는 용도로 사용된다.
  - 하나의 프로그램 내에서는 둘 이상의 소켓이 존재할 수 있으므로, 둘 이상의 PORT가 하나의 프로그램에 의해 할당될 수 있다.
  - PORT번호는 16비트로 표현, 따라서 그 값은 0 이상 65535 이하
  - 0~1023은 잘 알려진 PORT(Well-known PORT)라 해서 이미 용도가 결정되어 있다.



PORT번호에 의한 소켓의 구분과정

6주차 수업이 끝났습니다

고생하셨습니다.