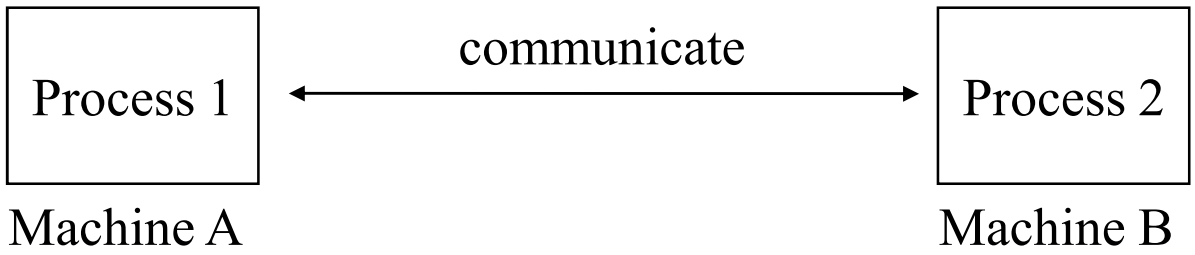
Network IPC: Sockets

IPC (Inter Process Communication)

- pipes, FIFOs, message queues, semaphores, and shared memory
  - allow processes running on the same machine to communicate with one another.
- socket
  - allows processes running on different machines to communicate with one another.



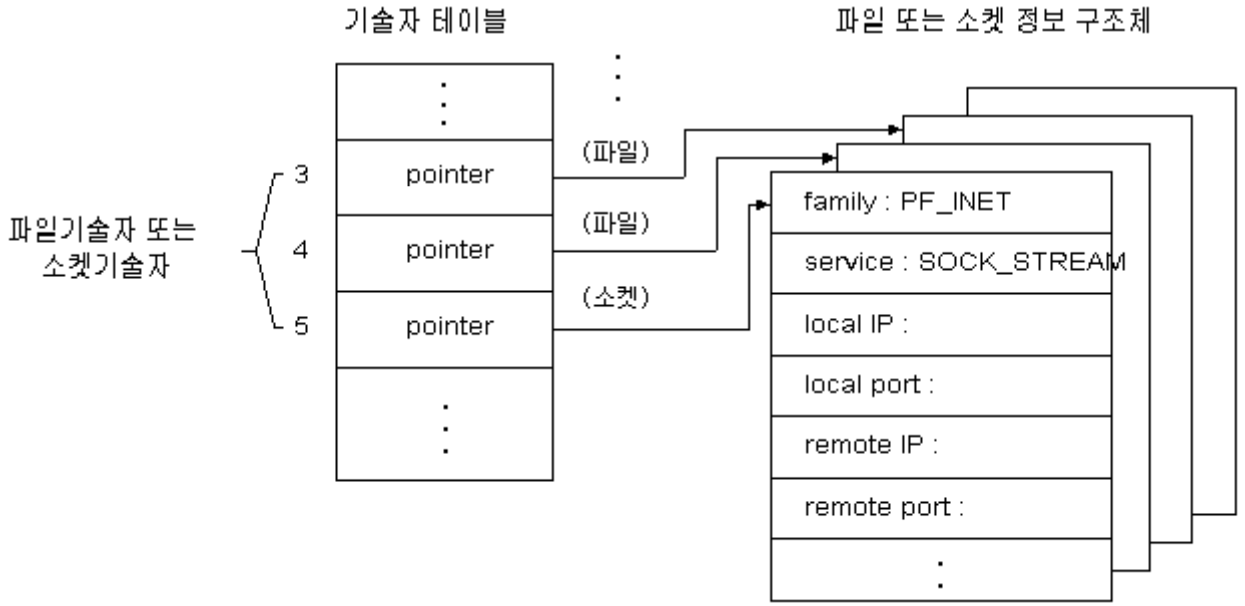
Socket

- Socket
  - an abstraction of a communication endpoint.

- Socket descriptor
  - Just as applications would use file descriptors to access a file, applications use socket descriptors to access sockets.
  - Socket descriptors are implemented as file descriptors.
    - many of the functions that deal with file descriptors (read and write) will work with a socket descriptor.
    - But, lseek doesn't work with sockets, since sockets don't support the concept of a file offset.

Socket

- Socket descriptor and file descriptor shares a table.
  - 3 and 4 are file descriptors.
  - 5 is a socket descriptor.



  - descriptor table은 per process structure이다. 따라서, 다른 process간에는 같은 값의 descriptor를 가질 수 있다.

socket()

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

int socket(int domain, int type, int protocol);
                                    Returns: file (socket) descriptor if OK, -1 on error
```

- Create a socket.
  - *domain* argument determines the nature of the communica tion, including the address format.
    - AF_INET: IPv4 Internet domain
      - Usually, this is used.
      - PF_INET is also used.
    - AF_INET6: IPv6 Internet domain
    - AF_UNIX: UNIX domain
    - AF_UNSPEC: unspecified

socket()

- *type* argument determines the type of the socket, which further determines the communication characteristics.
  - SOCK_DGRAM: connectionless, unreliable
  - SOCK_STREAM: connection-oriented, reliable
- *protocol* argument selects the default protocol for the given domain and socket type.
  - TCP (Transmission Control Protocol):
    - SOCK_STREAM in AF_INET domain
  - UDP (User Datagram Protocol):
    - SOCK_DGRAM in AF_INET domain
  - Usually, zero is used.

socket()

- example

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

int main()
{
        int fd1, fd2, sd1, sd2;

        fd1 = open("/etc/passwd", O_RDONLY);
        printf("/etc/passwd's file descriptor = %d\n", fd1);

        sd1 = socket(PF_INET, SOCK_STREAM, 0);
        printf("stream socket descriptor = %d\n", sd1);
```

socket()

- example(cont.)

```
        sd2 = socket(PF_INET, SOCK_DGRAM, 0);
        printf("datagram socket descriptor = %d\n", sd2);

        fd2 = open("/etc/hosts", O_RDONLY);
        printf("/etc/hosts's file descriptor = %d\n", fd2);

        close(fd2) ;
        close(fd1) ;
        close(sd2) ;
        close(sd1) ;
}
```

socket()

- 실 행

```
$gcc -o open_socket open_socket.c -lsocket
$./open_socket
/etc/passwd's file descriptor = 3
stream socket descriptor = 4
datagram socket descriptor = 5
/etc/hosts's file descriptor = 6
$
```

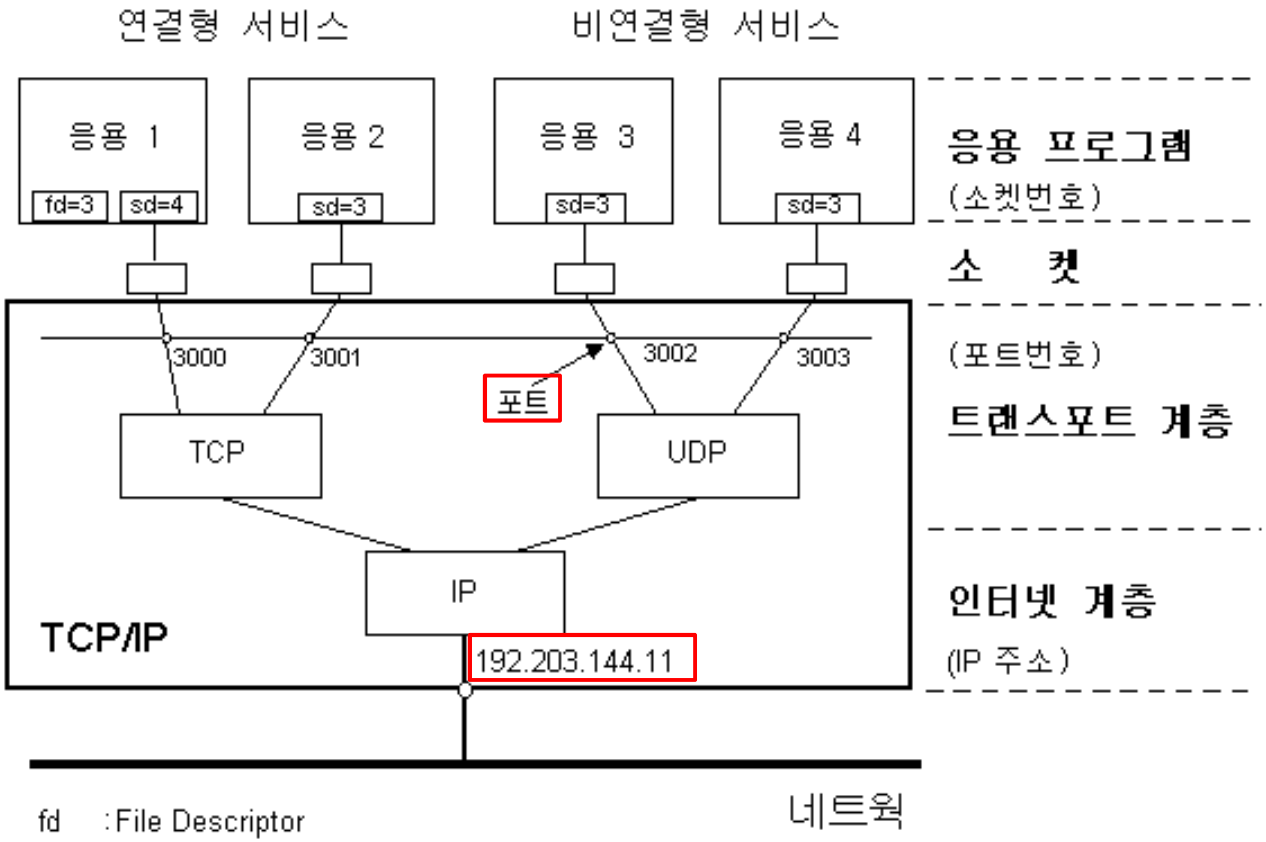
shutdown()

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

int shutdown (int sockfd, int how);
                                        Returns: 0 if OK, -1 on error
```

- disable I/O on a socket.
  - Is similar to close().
  - *how* argument
    - SHUT_RD: reading from socket is disabled.
    - SHUT_WR: transmitting data to socket is disabled.
    - SHUT_RDWR: both data transmission and reception are disabled.

Socket address

- Overall structure including socket



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

Socket address

Generic socket address structure

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

struct sockaddr {
        sa_family_t  sa_family;        /* address family */
        char         sa_data[];        /* variable-length address */
};
```

- socket을 이용할 통신 객체(client or server)의 주소를 표현하기 위해서는 address family, IP address, port number가 지정되어야 하며, 이 주소 정보를 socket address라고 부른다.
- 이 sockaddr 구조체에 IP address, port number 등을 직접 쓰거나 읽기가 불편하므로, sockaddr 구조체를 사용하는 대신 4 바이트의 IP address와 2 바이트의 port 번호를 구분하여 지정할 수 있는 인터넷 전용 소켓주소 구조체인 sockaddr_in을 주로 사용한다.

Socket address

- Internet protocol address structure

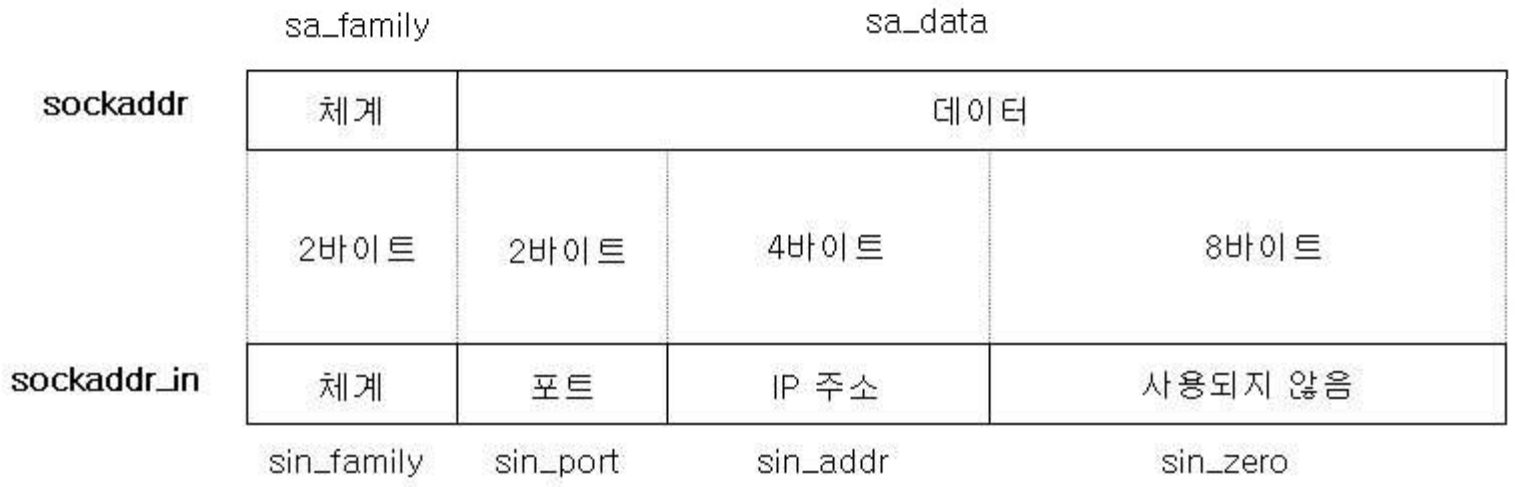
```
#include <netinet/in.h>

struct in_addr {
        in_addr_t      s_addr;          /* IPv4 address */
};

struct sockaddr_in {
        sa_family_t    sin_family;      /* address family */
        in_port_t      sin_port;        /* port number (2bytes) */
        struct in_addr sin_addr;        /* IPv4 address (4bytes) */
        char sin_zero[8];               /* not used */
};
```

  - sin_familly
    - AF_INET, AF_UNIX, …

Socket address

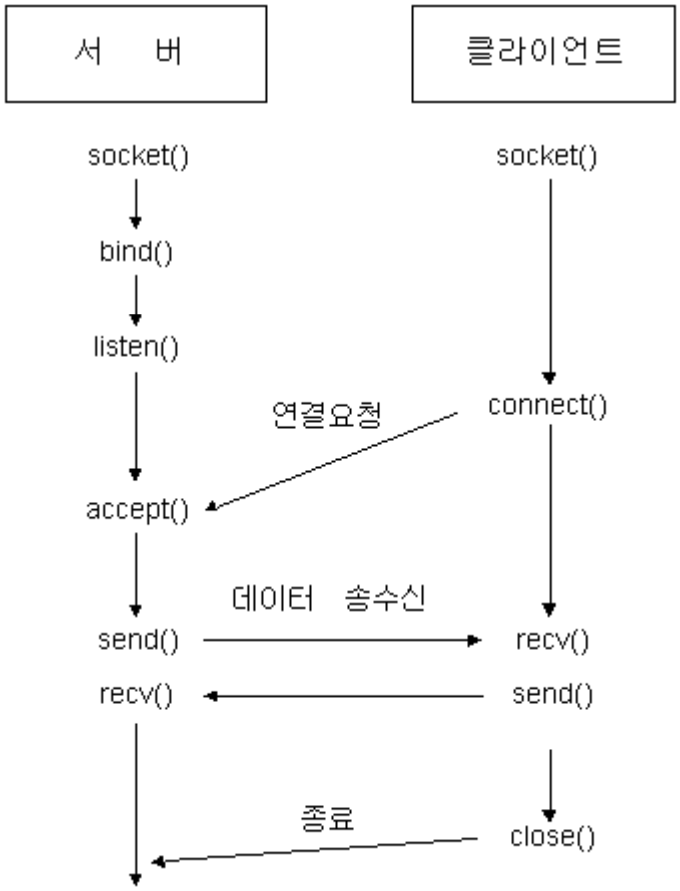
Socket address structure

- sockaddr_in은 sockaddr 구조체의 데이터를 internet protocol에서 사용하기에 적합하도록 수정한 것이다.



Socket programming

- socket programming's basic procedure



bind()

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

int bind(int sockfd, const struct sockaddr *addr, socklen_t len);
                                                    Returns: 0 if OK, -1 on error
```

- associate an address with a socket.
  - For a server, we need to associate a well-known address with the server's socket on which client requests will arrive.
  - For client, we can let the system choose a default address.
    - ➔ bind() is not used in client side.
  - *len* is the size of socket address.

connect()

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

int connect(int sockfd, const struct sockaddr *addr, socklen_t len);
                                                    Returns: 0 if OK, -1 on error
```

- Client requests a connection to server.
  - Before exchanging data, we need to create a connection between the socket of client and server.
  - *addr* is the address of the server.

listen()

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

int listen(int sockfd, int backlog);
                                        Returns: 0 if OK, -1 on error
```

- A server announces that it is willing to accept connect requests.
  - *backlog* specifies how many connection requests can be queued. (queue size)

accept()

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

int accept(int sockfd, struct sockaddr *addr, socklen_t *len);
                                        Returns: file (socket) descriptor if OK, -1 on error
```

- Server accept a connect request from clients.
  - *addr* is the socket address of client.
  - The file descriptor returned is a socket descriptor that is connected to the client that called connect().

send()

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

ssize_t send(int sockfd, const void *buf, size_t nbytes, int flags);
                                        Returns: number of bytes sent if OK, -1 on error
```

- Send data to other process.
  - is similar to write, but
  - allows us to specify flags to change how the data we want to transmit is treated
    - MSG_DONTROUTE, MSG_DONTWAIT, MSG_EOR, MSG_OOB

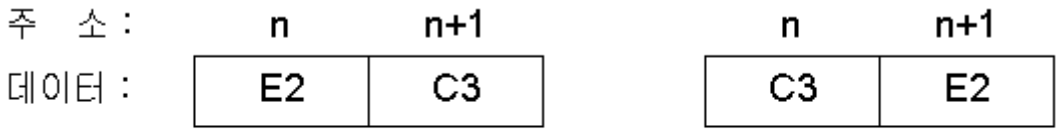
receive()

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

ssize_t recv(int sockfd, void *buf, size_t nbytes, int flags);
        Returns: length of message in bytes,
        0 if no messages are available and peer has done an orderly shutdown,
        or -1 on error
```

- Receive data from other process.
  - is similar to read, but
  - allows us to specify some options to control how we receive the data.
    - MSG_OOB, MSG_PEEK, MSG_TRUNC, MSG_WAITALL

Byte ordering

- Two types of byte order
  - Little endian
    - the least significant byte contains the lowest byte address
    - Intel, DEC
  - Big endian
    - the least significant byte contains the highest byte address
    - IBM, Motorola
  - Eg. 0xC3E2



(a) 80x86 계열 (b) MC68000계열

Byte ordering

- Byte ordering
  - Host byte order
    - 컴퓨터가 memory에 byte를 저장하는 순서
  - Network byte order
    - network에서 byte 단위로 data가 전달되는 순서
    - 0xC3E2의 경우 C3, E2의 순서로 전달 (like big endian)
  - 즉, intel의 80x86계열의 CPU가 사용하는 host byte order는 network byte order와 다르다. 따라서 80x86계열의 컴퓨터에서 network를 통하여 전송한 데이터를 68000계열의 컴퓨터가 수신하면 byte 순서가 바뀌게 된다.

Byte ordering

- Byte ordering
  - 이러한 문제를 해결하기 위하여 컴퓨터 내부에서 만들어진 host byte order 데이터를 network로 전송하기 전에 htons() 함수를 사용하여 모두 network byte order로 바꾸어야 한다.
  - 반대로 network에서 수신한 데이터는 ntohs() 함수를 사용하여 자신에게 맞는 host byte order로 바꾸어야 한다.
  - 즉, network byte order를 지켜 데이터를 전송함으로써 수신한 데이터가 어떤 종류의 컴퓨터에서 만들어진 것인지 알 필요가 없도록 하는 것이다.
  - Motolora 계열의 CPU에서는 host byte order와 network byte order가 같은데, 이러한 호스트에서의 htons()와 ntohs() 함수는 아무 일도 하지 않는다.

Byte ordering

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

uint32_t htonl(uint32_t hostint32);
uint16_t htons(uint16_t hostint16);
uint32_t ntohl(uint32_t netint32);
uint16_t ntohs(uint16_t netint16);
```

- convert between the host byte order and the network byte order.
  - htonl/htons (4byte/2byte)
    - Host byte order ➔ network byte order
  - ntohl/ntohs (4byte/2byte)
    - Network byte order ➔ host byte order

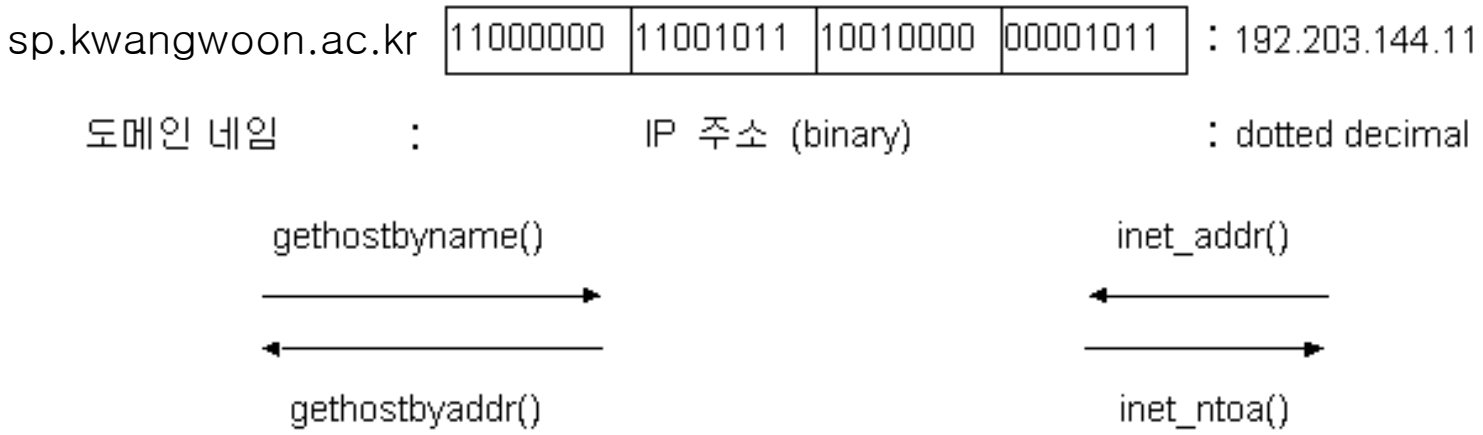
Address conversion

- IP address conversion
  - 32비트의 IP address를 편의에 따라 sp.kwangwoon.ac.kr과 같은 domain name, 그리고 192.203.144.11과 같은 dotted decimal 표시법 등으로 바꾸어 사용하고 있다.
  - 한편 IP packet을 network로 실제로 전송할 때에는 32비트의 IP address가 필요하다.
  - 따라서, 이들 주소 표현법을 자유롭게 변환할 수 있는 함수가 필요하다.

Address conversion

IP address conversion

- 예: 아래 그림에서 dotted decimal로 표현된 192. 203.144.11을 32 비트의 IP address로 변환하려면 inet_addr() 시스템 콜을 사용하고 IP address를 다시 dotted decimal로 변환하려면 inet_ntoa()를 사용한다.



Address conversion

- example

```
struct sockaddr_in server_addr;
server_addr.sin_addr.s_addr = inet_addr ("192.203.144.11");
printf("%x\n", server_addr.sin_addr.s_addr);          /* hexa 4바이트 출력 */
printf("%s\n", inet_ntoa(server_addr.sin_addr));      /* dotted decimal 출력 */
```

Server example

- Example

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

#define PORT    5555

int main(void)
{
        char buf[256];
        struct sockaddr_in server, client;
        int sd, cd, clientlen = sizeof(client);
```

Server example

- Example(cont.)

```
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
}
memset((char *)&server, '\0', sizeof(server));
server.sin_family               = AF_INET;
server.sin_addr.s_addr          = inet_addr("127.0.0.1");
server.sin_port                 = htons(PORT);

if(bind(sd, (struct sockaddr*)&server, sizeof(server))) {
        perror("bind");
        exit(1);
}
if(listen(sd, 5)) {
        perror("listen");
        exit(1);
}
```

Server example

- Example(cont.)

```
        if ((cd = accept(sd, (struct sockaddr*)&client, &clientlen)) == -1) {
                perror("accept");
                exit(1);
        }

        sprintf(buf, "Your IP address is %s", inet_ntoa(client.sin_addr));
        if(send(cd, buf, strlen(buf) + 1, 0) == -1) {
                perror("send");
                exit(1);
        }
        close(cd);
        close(sd);

        return 0;
}
```

Client example

- Example

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

#define PORT    5555

int main(void)
{
        int sd;
        char buf[256];
        struct sockaddr_in server;
```

Client example

- Example(cont.)

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

memset((char *)&server, '\0', sizeof(server));
server.sin_family               = AF_INET;
server.sin_addr.s_addr          = inet_addr("127.0.0.1");
server.sin_port                 = htons(PORT);

if(connect(sd, (struct sockaddr*)&server, sizeof(server))) {
        perror("connect");
        exit(1);
}
```

Client example

- Example(cont.)

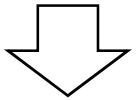
```
        if(recv(sd, buf, sizeof(buf), 0) == -1) {
                perror("recv");
                exit(1);
        }
        close(sd);
        printf("From Server: %s\n", buf);

        return 0;
}
```

Server & client example

Server 실행

```
$ ./test_server
```

(1) Server부터 실행



```
$ ./test_server
$
```

(3) Server 종료

Client 실행

```
$ ./test_client
From Server: Your IP address is 127.0.0.1
$
```

(2) Client 실행

Reference

- More about network programming
  - UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications, Prentice Hall, 1999.
  - UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998