Computer Networks (Lab)

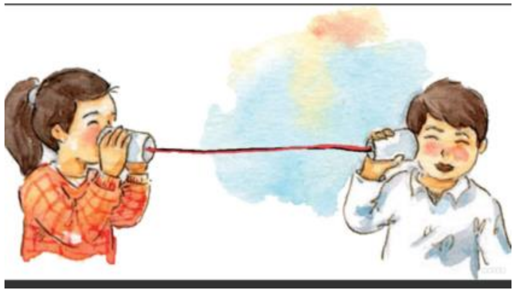
-Socket Basic-

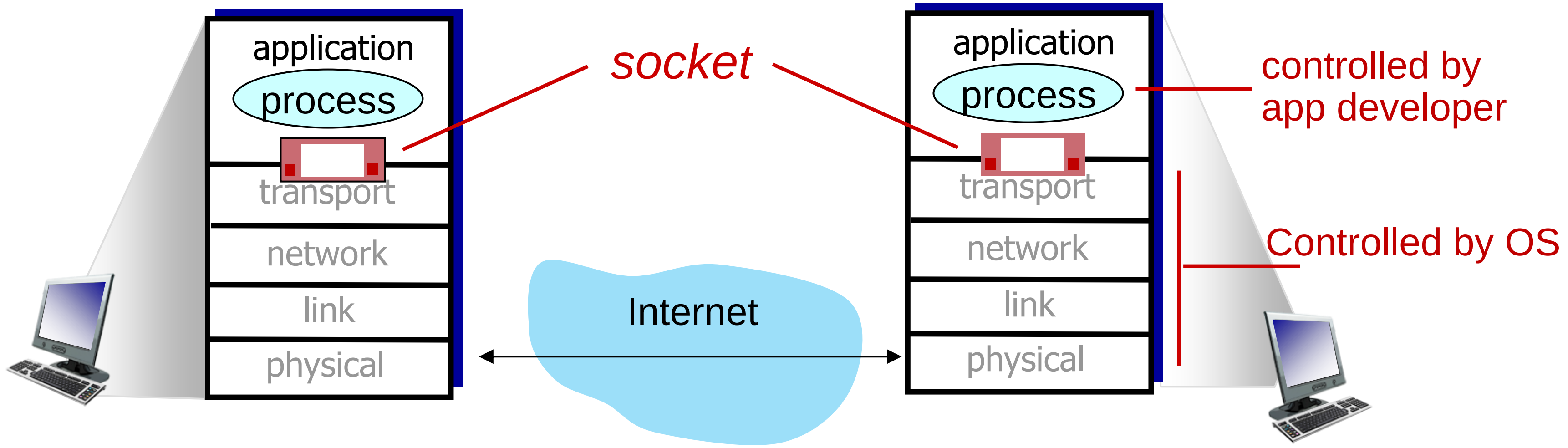
2025. 4. 2

Young Deok Park (박영덕)

Network Programming (Socket Programming)

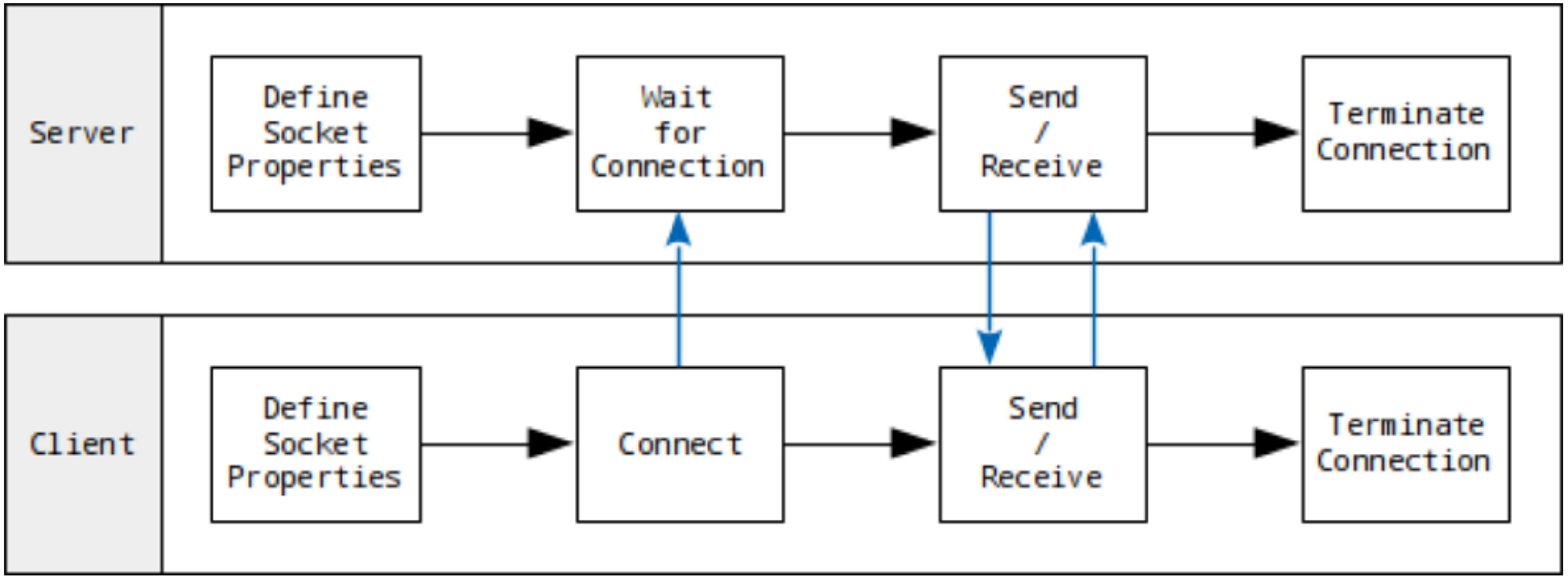
- Programming for communications among computers
- Socket
  - Process sends/receives messages to/from its socket
  - Socket analogous to door
    - Sending process shoves message out door
  - Two sockets involved: one on each side





Yeungnam University

Communication between Server & Client (Code's point of view)



Important Functions at Server Side

Define server socket

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

➜ 성공 시 파일 디스크립터, 실패 시 −1 반환

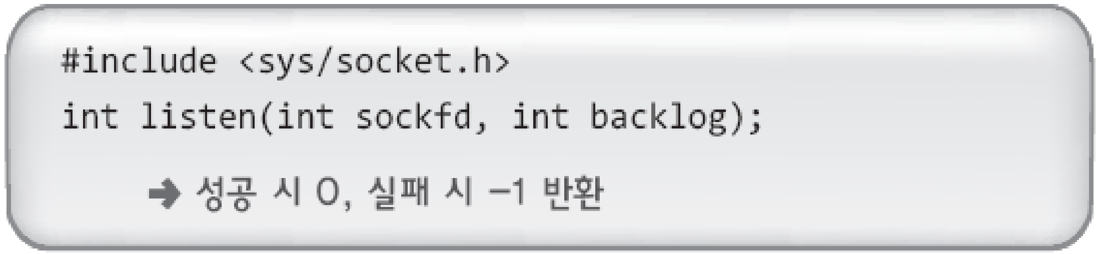
Binding (assign IP address and port # to the socket)

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

➜ 성공 시 0, 실패 시 −1 반환

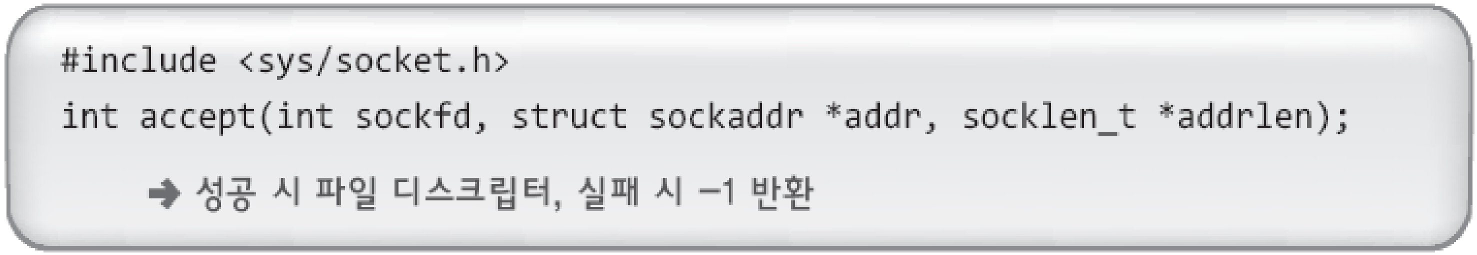
Important Functions at Server Side

Waiting for connection request from client



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

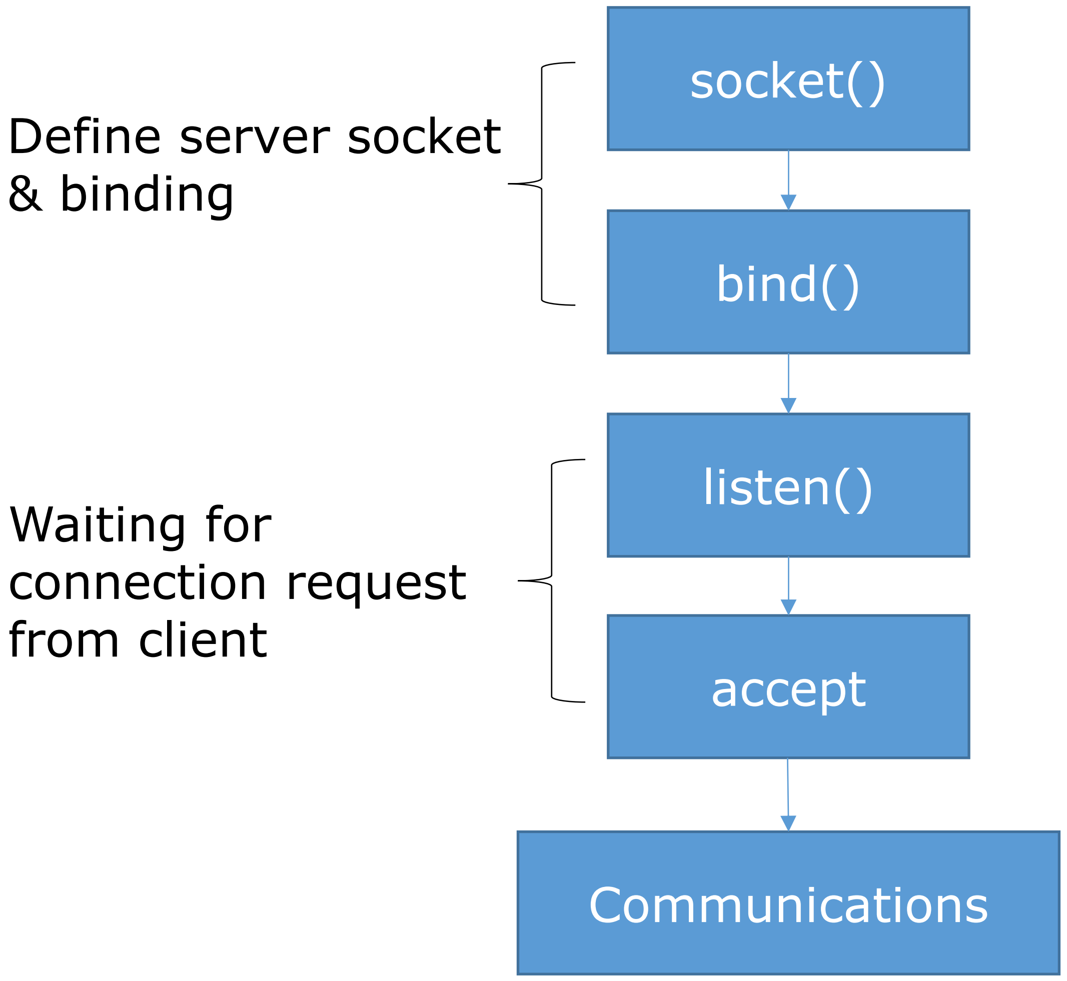
➜ 성공 시 0, 실패 시 -1 반환



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

➜ 성공 시 파일 디스크립터, 실패 시 -1 반환

Function Flow at Server Side


Define server socket & binding
socket()
bind()
Waiting for connection request from client
listen()
accept
Communications

Important Functions at Client

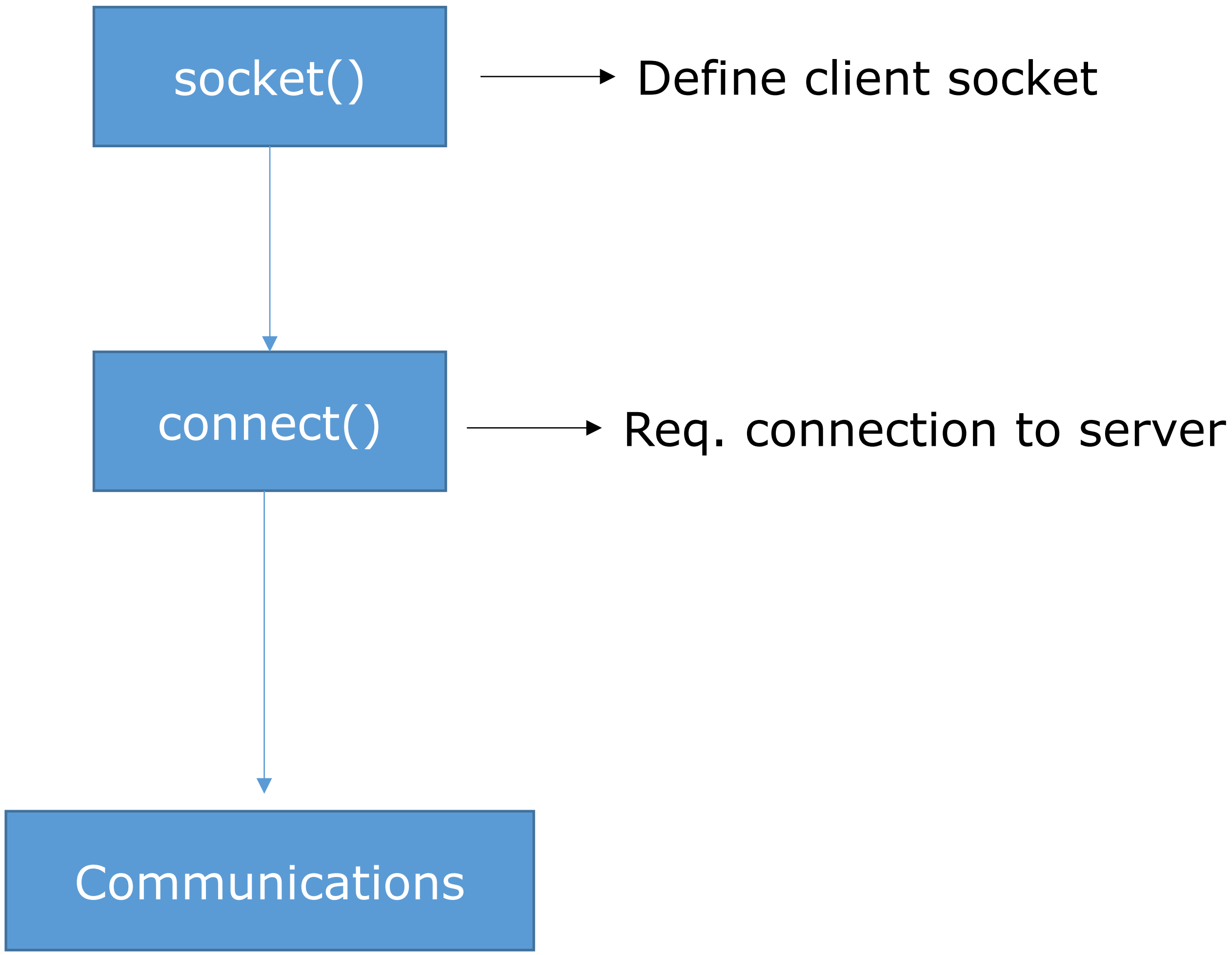
Define socket
(This function is used in both server and client sides)

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

    ➜ 성공 시 파일 디스크립터, 실패 시 -1 반환
```

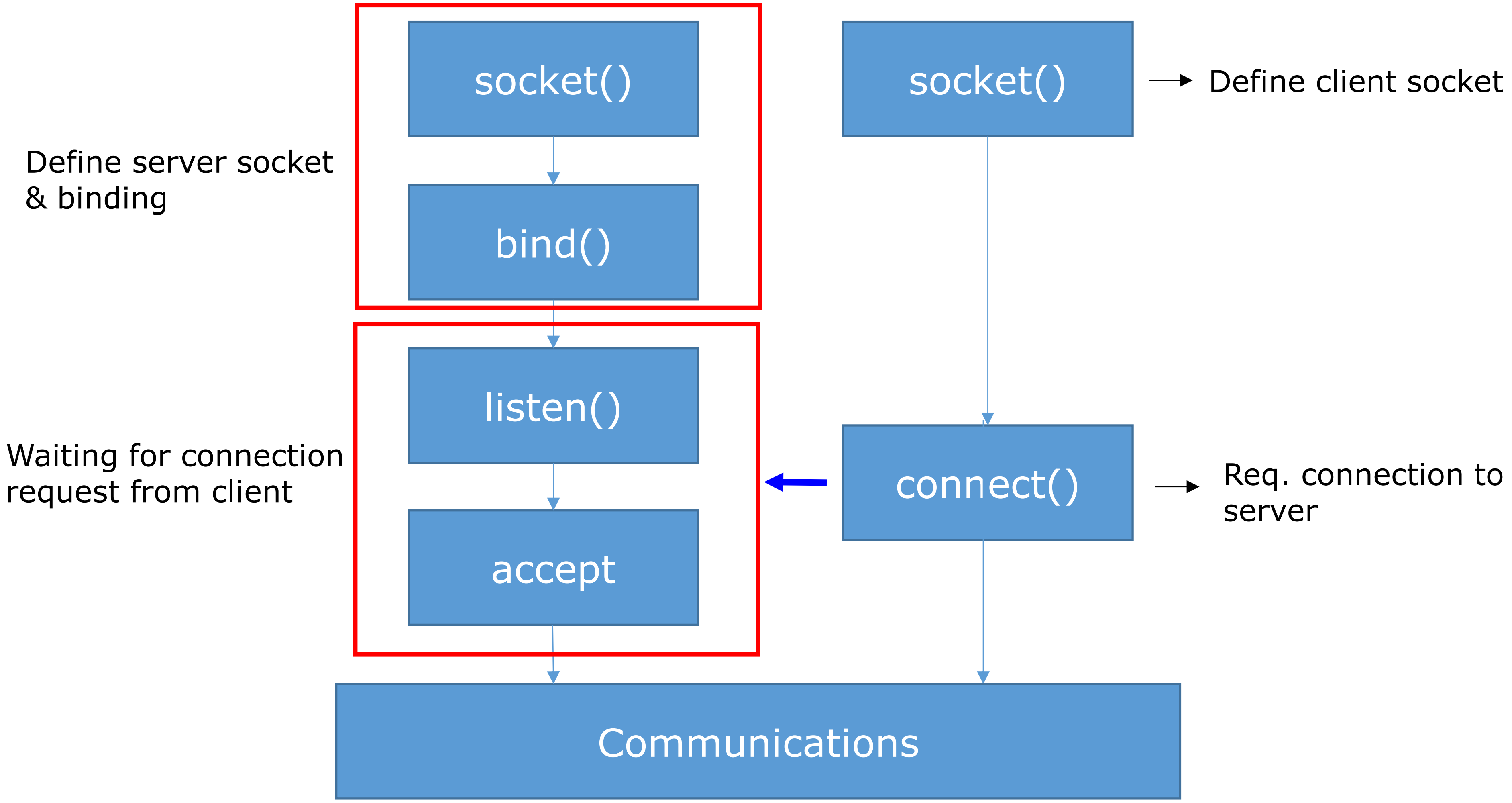
Request Connection to Server

```
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr *serv_addr, socklen_t addrlen);

    ➜ 성공 시 0, 실패 시 -1 반환
```

Function Flow at Client Side



Interaction Between Server & Client



socket()
bind()
listen()
accept()
socket()
connect()
Communications
Define server socket & binding
Waiting for connection request from client
Define client socket
Req. connection to server

Simple Server

```
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);
        }
```

Create Socket

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

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

Set IP ADDR, Port info.

```
        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");
```

Simple Server (cont'd)

```
        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);
}
```

Waiting for Connection Req. from client

Sending Message to client

Simple Client

```
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 <IP> <port>₩n", argv[0]);
                exit(1);
        }
                                                Create Socket
        sock=socket(PF_INET, SOCK_STREAM, 0);
        if(sock == -1)
                error_handling("socket() error");
```

Simple Client (cont'd)

```
//server side
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);
```

```
        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!");
```

Connection Req. to server

```
        str_len=read(sock, message, sizeof(message)-1);
        if(str_len==-1)
                error_handling("read() error!");
```

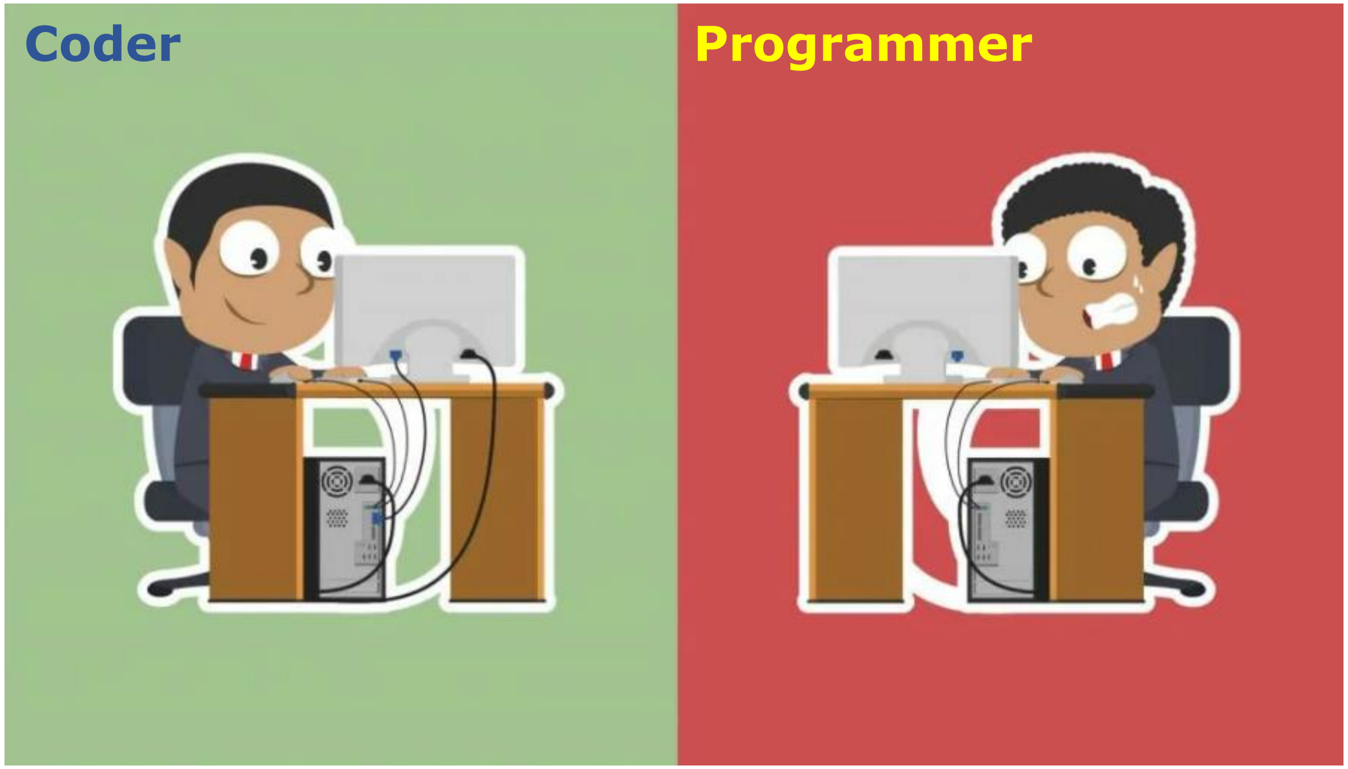
Receiving message from server

```
//server side
write(clnt_sock, message, sizeof(message));
```

```
        printf("Message from server: %s ₩n", message);
        close(sock);
        return 0;
}
```

Exercise!

- Now, we are just coders, not programmers!!



Execute!

- Q: Which program should be run first?
  - A: server

- ./simple_server port#

```
$ gcc hello_server.c –o simple_server
$ ./simple_server  7777
```

- ./simple_client ip address (127.0.0.1 → Loopback) port#

```
$ gcc hello_client.c –o simple_client
$ ./simple_client  127.0.0.1 7777
```