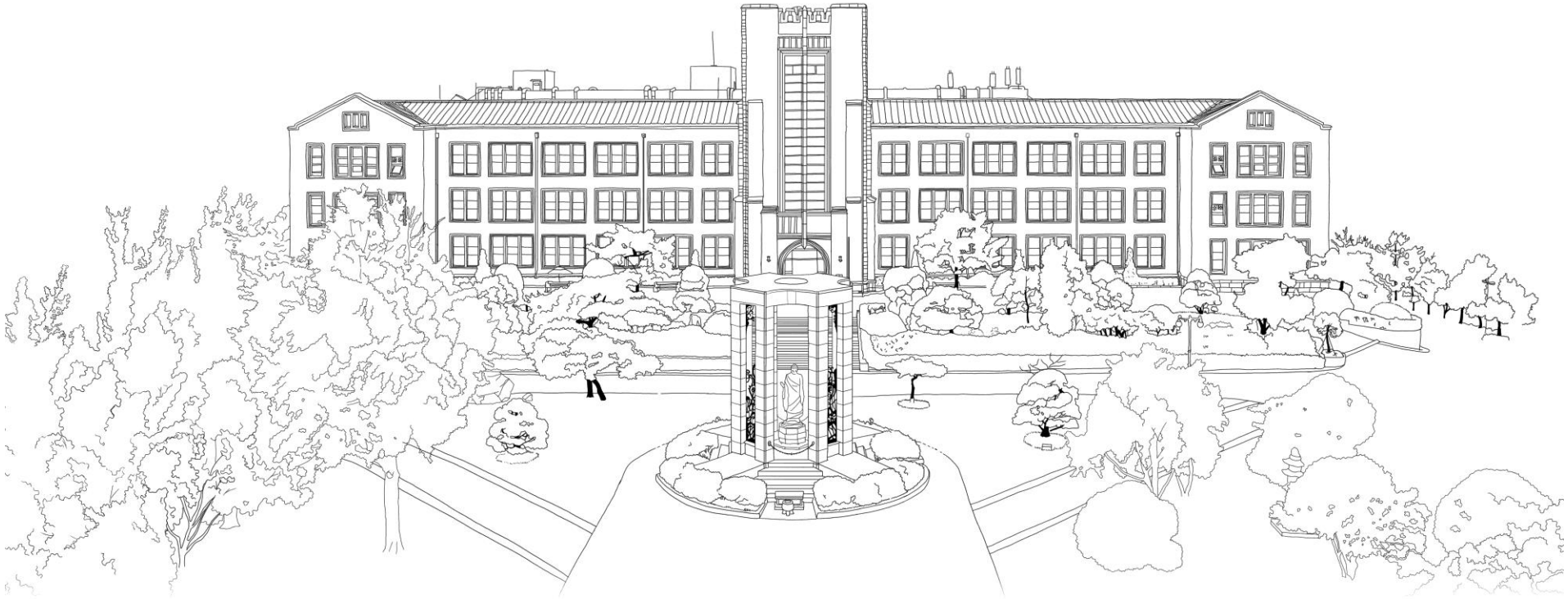


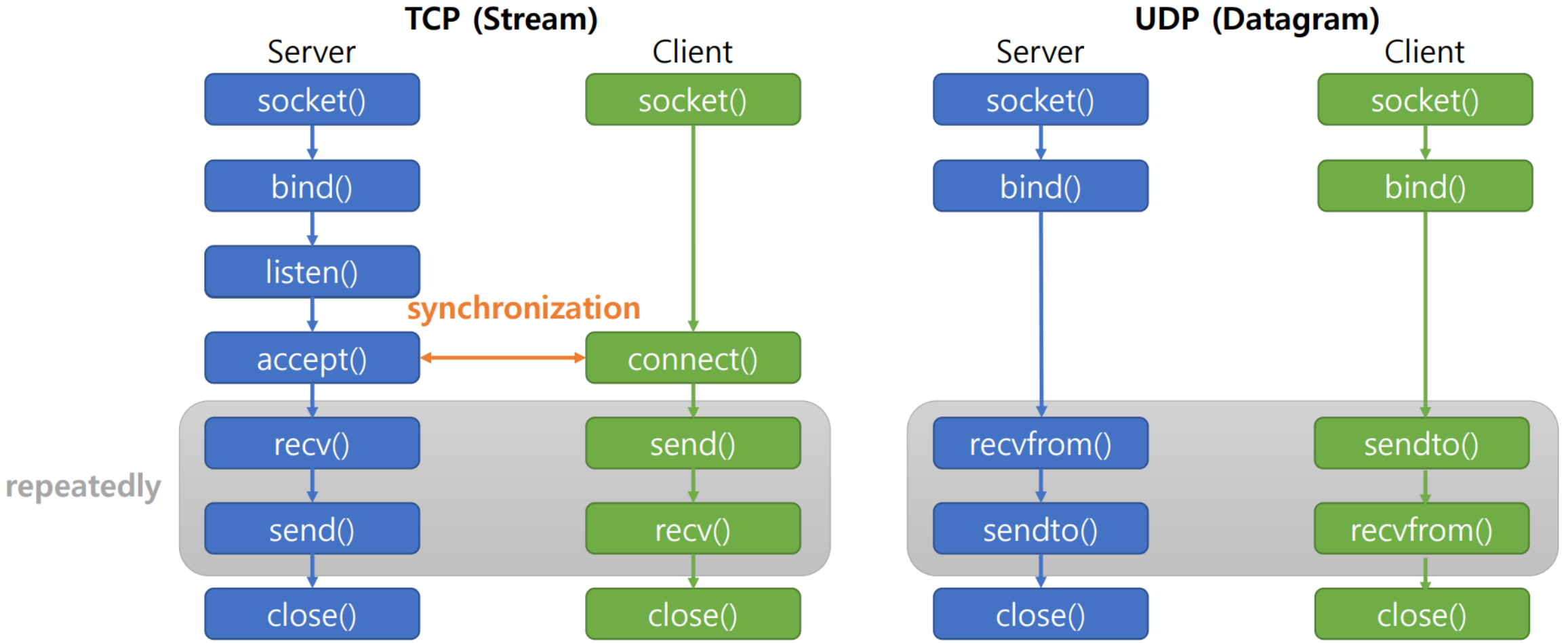
2021-02

멀티미디어 네트워크 실습 1

BigDataLab
Dept. of Multimedia Engineering at Dongguk University

dongguk
UNIVERSITY

❖ Client – Server Communication



❖ TCP/IP

- TCP/IP 주소체계를 위한 C언어의 socket 구조체
- sockaddr_in 구조체는 sockaddr 구조체로 casting 가능
  - ➢ struct sockaddr_in server_addr;
  - ➢ (struct sockaddr*)&server_addr;

```
struct sockaddr {                       /* generic structure */
        unsigned short sa_family;       /* Address family */
        char sa_data[14];               /* Family-specific address information */
}

struct in_addr {
        unsigned long s_addr;           /* Internet address (32 bits) */
}

struct sockaddr_in {
        unsigned short sin_family;      /* Internet protocol (AF_INET): Address Family*/
        unsigned short sin_port;        /* Address port (16 bits) */
        struct in_addr sin_addr;        /* Internet address (32 bits) */
        chart sin_zero[8];              /* Not used */
}
```

❖ TCP/IP (Server / Client)

- socket() : 서버 또는 클라이언트의 socket 생성을 위한 함수
- return 값은 integer이며 음수는 socket 생성 실패를 의미

```
int sock_id = socket(family, type, protocol);
sock_id (int):      socket descriptor
family (int):       PF_INET (IPv4 protocol, Internet Addresses), PF_UNIX(File addresses)
type:               SOCK_STREAM(reliable, connection-based service),
                    SOCK_DGRAM(unreliable, connectionless)
protocol:           IPPROTO_TCP
                    IPPROTO_UDP
```

❖ TCP/IP (Server)

- listen() : 클라이언트가 통신 요청할 수 있도록 socket을 대기 상태로 만들고 클라이언트 연결 큐를 생성하는 함수
- 최대 queue_limit 값만큼 연결 요청을 허용
- return 값은 integer이며 -1는 연결 대기 실패를 의미

```
int status = listen(sock_id, queue_limit);
sock_id (int):          socket descriptor
queue_limit (int):      # of active participants that can "wait" for a connection
```

❖ TCP/IP (Server)

- accept() : 클라이언트의 통신 요청을 기다리는 함수
- 클라이언트의 통신 요청이 오면 클라이언트와 데이터 송수신을 위한 socket 번호 생성
- blocking : 클라이언트의 통신 요청(connection()이 있을 때까지 대기)
- return 값은 integer이며 음수는 클라이언트와의 연결 실패를 의미

```
int status = accept(sock_id, &client_addr, &addr_len);
sock_id (int):                  socket descriptor
client_addr (struct sockaddr):  address of the active participant
addr_len (bytes):               sizeof(client_addr)
```

❖ TCP/IP (Server / Client)

- write() : 클라이언트 또는 서버에게 메시지를 송신하는 함수 (send())
- read() : 클라이언트 또는 서버에게 메시지를 수신하는 함수 (recv())
- blocking : 메시지를 전달한 후에 함수 리턴

```
write(sock_id, msg, msg_len);
sock_id (int):              socket descriptor
msg (const void[]):         message to be transmitted
addr_len (bytes):           length of message(msg) to transmit

read(sock_id, recv_buf, buf_len);
sock_id (int):              socket descriptor
recv_buf (void[]):          stores received bytes
buf_len (bytes):            length of received message(recv_buf)
```

❖ TCP/IP (Server / Client)

- close() : socket 사용을 종료하는 함수
- return 값이 0이면 정상적인 socket 종료를 의미

```
close(sock_id);
sock_id (int):          socket descriptor
```

❖ TCP/IP (Server)

```
if((sock_flag = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0){
        printf("Socket 생성  실패 ...\n");
        exit(0);
}
else
        printf("Socket 생성  성공 ...\n");
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

❖ TCP/IP (Server)

```
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(PORT);

if((bind(sock_flag, (struct sockaddr*)&server_addr, sizeof(server_addr))) != 0){
        printf("Socket 바 인 딩  실 패 ...\n");
        exit(0);
}
else
        printf("Socket 바 인 딩  성 공 ...\n");
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

❖ TCP/IP (Server)

```
if((listen(sock_flag, CLIENT_SIZE)) != 0){
        printf("연 결  대 기  실 패 ...\n");
        exit(0);
}
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

Socket Programming

❖ TCP/IP (Server)

```
if((conn_flag = accept(sock_flag, (struct sockaddr*)&client_addr, &length)) < 0){
        printf("Server - Client 연결  실패 \n");
        exit(0);
}
else
        printf("Server - Client 연결  성공 \n");
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

❖ TCP/IP (Server)

```
void DATA_SEND_RECV(int sock_flag)
{
        char buf[MAX];

        //while(true){...}
        for(;;){
                memset(buf, 0x00, MAX);
                read(sock_flag, buf, sizeof(buf));
                printf("From Client : %s\nTo Client : ", buf);

                memset(buf, 0x00, MAX);
                fgets(buf, MAX, stdin);

                write(sock_flag, buf, sizeof(buf));

                if(strcmp("exit\n", buf) == 0){
                        printf("Server 종료...\n");
                        break;
                }
        }
}
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

❖ TCP/IP (Server)

```
close(conn_flag);
close(sock_flag);
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

❖ TCP/IP (Client)

```
if((sock_flag = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0){
        printf("Socket 생성  실패 ...\n");
        exit(0);
}
else
        printf("Socket 생성  성공 ...\n");
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

❖ TCP/IP (Client)

```
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
server_addr.sin_port = htons(SERVER_PORT);

if((connect(sock_flag, (struct sockaddr*)&server_addr, sizeof(server_addr))) < 0){
        printf("Server - Client 연결  실패 \n");
        exit(0);
}
else
        printf("Server - Client 연결  성공 \n");
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

❖ TCP/IP (Client)

```
void DATA_SEND_RECV(int sock_flag)
{
        char buf[MAX];

        //while(true){...}
        for(;;){
                memset(buf, 0x00, MAX);
                printf("Enter the string : ");
                fgets(buf, MAX, stdin);
                write(sock_flag, buf, sizeof(buf));

                if(strcmp("exit\n", buf) == 0){
                        printf("Client 종료 ...\n");
                        break;
                }

                memset(buf, 0x00, MAX);
                read(sock_flag, buf, sizeof(buf));

                printf("From Server : %s\n", buf);
                if(strcmp("exit\n", buf) == 0){
                        printf("Server 종료로 인한 Client 종료 ...\n");
                        break;
                }

        }
}
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

❖ TCP/IP (Client)

```
close(sock_flag);
```

Server

1. 서버 TCP socket 생성: socket()
2. 서버 TCP socket 포트 할당: bind()
3. 클라이언트 연결 요청 대기: listen()
4. 클라이언트 연결 수락: accept()
5. 데이터 송·수신: send() / recv()
6. socket 연결 종료: close()

Client

1. 클라이언트 TCP socket 생성: socket()
2. 서버에게 socket 연결 요청: connect()
3. 데이터 송·수신: send() / recv()
4. socket 연결 종료: close()

❖ TCP/IP (Server)

```c
#include <stdio.h>
#include <sys/socket.h> //socket
#include <netinet/in.h> //IPPROTO_TCP, sockaddr_in
#include <arpa/inet.h>  //inet_addr
#include <stdlib.h>     //implict declaration of function 'exit'
#include <string.h>     //memset()
#include <unistd.h>

#define MAX 128
#define PORT 8002
#define CLIENT_SIZE 1

void DATA_SEND_RECV(int sock_flag);

int main(int argc, char*argv[])
{
        int sock_flag, conn_flag, length;
        struct sockaddr_in server_addr, client_addr;

        if((sock_flag = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0){
                printf("Socket 생성  실패 ...\n");
                exit(0);
        }
        else
                printf("Socket 생성  성공 ...\n");

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        server_addr.sin_port = htons(PORT);

        if((bind(sock_flag, (struct sockaddr*)&server_addr, sizeof(server_addr))) != 0){
                printf("Socket 바인딩  실패 ...\n");
                exit(0);
        }
        else
                printf("Socket 바인딩  성공 ...\n");

        if((listen(sock_flag, CLIENT_SIZE)) != 0){
                printf("연결  대기  실패 ...\n");
                exit(0);
        }

        length = sizeof(client_addr);

        if((conn_flag = accept(sock_flag, (struct sockaddr*)&client_addr, &length)) < 0){
                printf("Server - Client 연결  실패 \n");
                exit(0);
        }
        else
                printf("Server - Client 연결  성공 \n");

        DATA_SEND_RECV(conn_flag);


        close(conn_flag);
        close(sock_flag);

        return 0;
}
```

```c
void DATA_SEND_RECV(int sock_flag)
{
        char buf[MAX];

        //while(true){...}
        for(;;){
                memset(buf, 0x00, MAX);
                read(sock_flag, buf, sizeof(buf));
                printf("From Client : %s\nTo Client : ", buf);

                memset(buf, 0x00, MAX);
                fgets(buf, MAX, stdin);

                write(sock_flag, buf, sizeof(buf));

                if(strcmp("exit\n", buf) == 0){
                        printf("Server 종 료 ...\n");
                        break;
                }
        }
}
```

❖ TCP/IP (Client)

```c
#include <stdio.h>
#include <sys/socket.h> //socket
#include <netinet/in.h> //IPPROTO_TCP, sockaddr_in
#include <arpa/inet.h>  //inet_addr
#include <stdlib.h>     //implict declaration of function 'exit'
#include <string.h>     //memset()
#include <unistd.h>

#define MAX 128
#define SERVER_PORT 8002

void DATA_SEND_RECV(int sock_flag);

int main(int argc, char*argv[])
{
        int sock_flag;
        struct sockaddr_in server_addr;

        if((sock_flag = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0){
                printf("Socket 생성  실패 ...\n");
                exit(0);
        }
        else
                printf("Socket 생성  성공 ...\n");

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(SERVER_PORT);

        if((connect(sock_flag, (struct sockaddr*)&server_addr, sizeof(server_addr))) < 0){
                printf("Server - Client 연결  실패 \n");
                exit(0);
        }
        else
                printf("Server - Client 연결  성공 \n");

        DATA_SEND_RECV(sock_flag);


        close(sock_flag);


        return 0;
}
```

```c
void DATA_SEND_RECV(int sock_flag)
{
        char buf[MAX];

        //while(true){...}
        for(;;){
                memset(buf, 0x00, MAX);
                printf("Enter the string : ");
                fgets(buf, MAX, stdin);
                write(sock_flag, buf, sizeof(buf));

                if(strcmp("exit\n", buf) == 0){
                        printf("Client 종료 ...\n");
                        break;
                }

                memset(buf, 0x00, MAX);
                read(sock_flag, buf, sizeof(buf));

                printf("From Server : %s\n", buf);
                if(strcmp("exit\n", buf) == 0){
                        printf("Server 종료로  인한  Client 종료 ...\n");
                        break;
                }

        }
}
```

❖ UDP (Server)

```c
#include <stdio.h>
#include <sys/socket.h> //socket
#include <netinet/in.h> //IPPROTO_UDP, sockaddr_in
#include <arpa/inet.h>  //inet_addr
#include <stdlib.h>     //implict declaration of function 'exit'
#include <string.h>     //memset()
#include <unistd.h>

#define MAX 128
#define PORT 8081

int main(int argc, char*argv[])
{
        int sock_flag;
        char buf[MAX];

        struct sockaddr_in server_addr, client_addr;
        int addrlen = sizeof(client_addr);

        if((sock_flag = socket(PF_INET, SOCK_DGRAM, 0)) < 0){
                printf("Socket 생성  실패 ...\n");
                exit(0);
        }
        else
                printf("Socket 생성  성공 ...\n");

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        server_addr.sin_port = htons(PORT);

        if((bind(sock_flag, (struct sockaddr*)&server_addr, sizeof(server_addr))) != 0){
                printf("Socket 바인딩  실패 ...\n");
                exit(0);
        }
        else
                printf("Socket 바인딩  성공 ...\n");
```

```c
        //while(true){...}
         for(;;){
                memset(buf, 0x00, MAX);

                recvfrom(sock_flag, buf, sizeof(buf), MSG_WAITALL, (struct sockaddr *)&client_addr, &addrlen);
                printf("From Client : %s\nTo Client : ", buf);

                memset(buf, 0x00, MAX);
                fgets(buf, MAX, stdin);

                sendto(sock_flag, buf, sizeof(buf), MSG_CONFIRM, (struct sockaddr *)&client_addr, addrlen);
                if(strcmp("exit\n", buf) == 0){
                        printf("Server 종료 ...\n");
                        break;
                }
        }

        close(sock_flag);

        return 0;
}
```

❖ UDP (Client)

```c
#include <stdio.h>
#include <sys/socket.h> //socket
#include <netinet/in.h> //IPPROTO_UDP, sockaddr_in
#include <arpa/inet.h>  //inet_addr
#include <stdlib.h>     //implict declaration of function 'exit'
#include <string.h>     //memset()
#include <unistd.h>

#define MAX 128
#define SERVER_PORT 8081

int main(int argc, char*argv[])
{
        int sock_flag;
        char buf[MAX];

        struct sockaddr_in server_addr;
        int addrlen = sizeof(server_addr);

        if((sock_flag = socket(PF_INET, SOCK_DGRAM, 0)) < 0){
                printf("Socket 생성  실패 ...\n");
                exit(0);
        }
        else
                printf("Socket 생성  성공 ...\n");

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(SERVER_PORT);
```

```c
        //while(true){...}
        for(;;){
                memset(buf, 0x00, MAX);
                printf("Enter the string : ");
                fgets(buf, MAX, stdin);

                sendto(sock_flag, buf, sizeof(buf), MSG_CONFIRM, (struct sockaddr *)&server_addr, addrlen);

                if(strcmp("exit\n", buf) == 0){
                        printf("Client 종료 ...\n");
                        break;
                }

                memset(buf, 0x00, MAX);

                recvfrom(sock_flag, buf, sizeof(buf), MSG_WAITALL, (struct sockaddr *)&server_addr, &addrlen);

                printf("From Server : %s\n", buf);
                if(strcmp("exit\n", buf) == 0){
                        printf("Server 종료로  인한  Client 종료 ...\n");
                        break;
                }

        }

        close(sock_flag);

        return 0;
}
```

Thank you!