컴퓨터 네트워크

- 소켓 프로그래밍 -

순천향대학교 사물인터넷학과

애플리케이션 계층

2.1 네트워크 애플리케이션의 원리

2.2 웹과 HTTP

2.3 전자메일

- SMTP, POP3, IMAP

2.4 DNS - 인터넷 디렉토리 서비스

2.5 P2P 파일 분배

2.6 비디오 스트리밍과 컨텐츠 분배 네트워크

2.7 소켓 프로그래밍: 네트워크 애플리케이션 작성

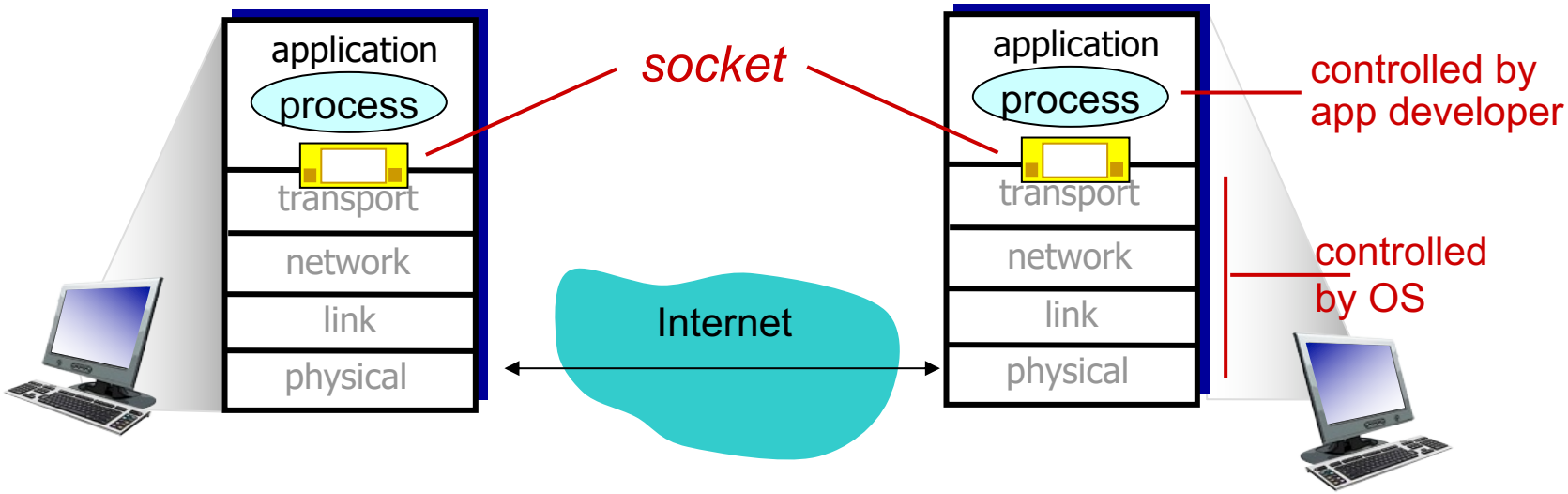
소켓 프로그래밍 Socket Programming

■ 목표

- 소켓을 이용한 클라이언트/서버 애플리케이션 프로그램 작성 방법 소개

■ 소켓Socket

- 애플리케이션 프로세스와 종단 간 트랜스포트 프로토콜 사이의 인터페이스



소켓 프로그래밍

- 2가지 트랜스포트 서비스에 대한 2가지 소켓 타입
  - UDP: 비연결형, 비신뢰적 데이터 전송
  - TCP: 연결형, 신뢰적인 데이터 전송

- 애플리케이션 프로그램 예제

1. 클라이언트는 키보드에서 1줄의 문자(데이터)를 읽어 들여, 서버로 전송
2. 서버는 데이터를 수신하여, 대문자로 변환
3. 서버는 수정된 데이터를 클라이언트로 전송
4. 클라이언트는 수정된 데이터를 수신하여 화면에 출력

UDP 소켓 프로그래밍

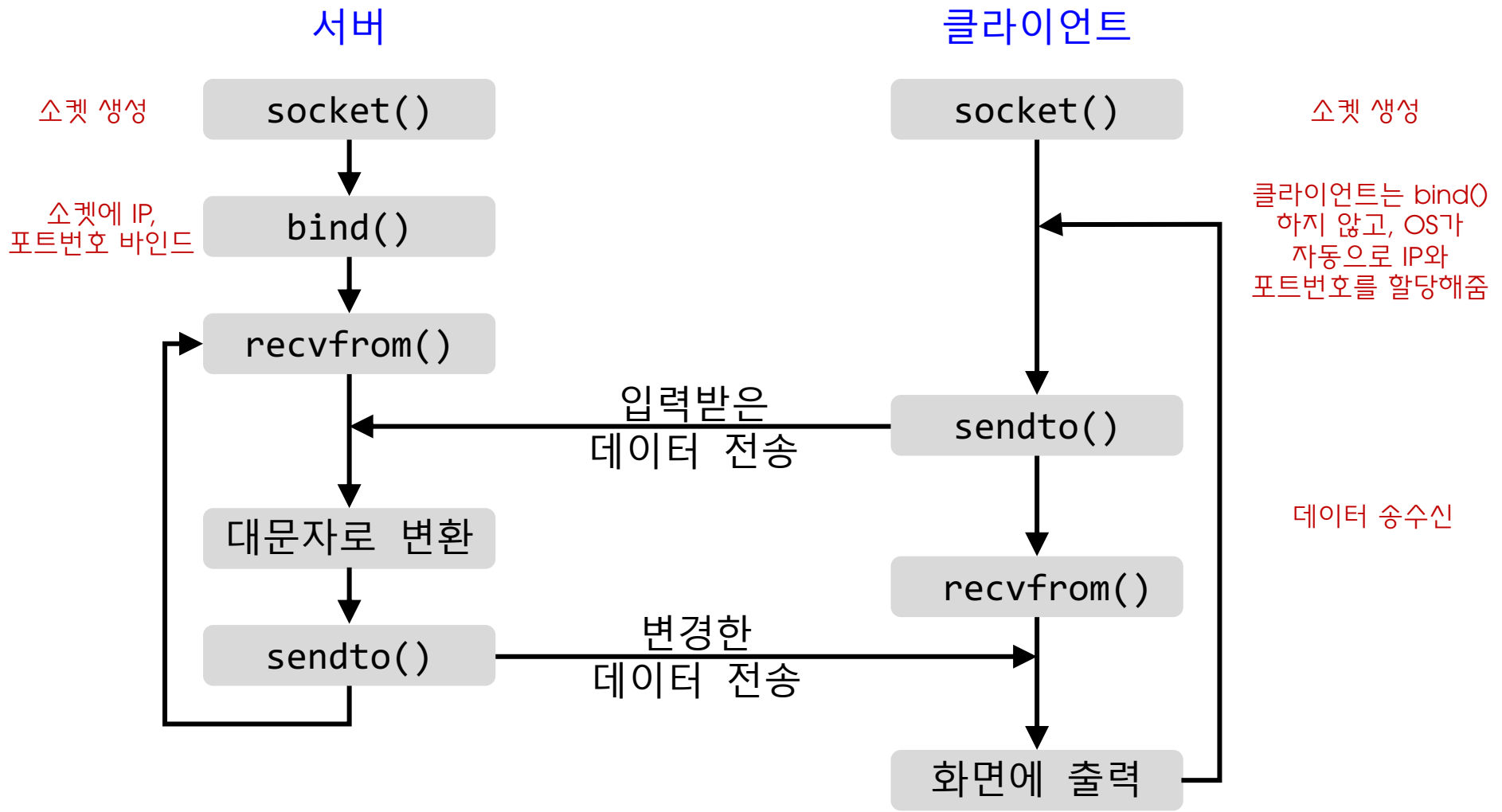
- UDP는 클라이언트와 서버 사이에 연결이 필요 없음
  - 데이터 전송 전, 핸드셰이킹 과정 없음
  - 송신 프로세스는 (연결이 없으므로) 각 패킷 마다 명시적으로 수신 프로세스의 IP 주소와 포트 번호를 붙임
  - 수신 프로세스는 수신된 패킷에서 송신 프로세스의 IP 주소와 포트 번호를 추출

- UDP는 비신뢰적 전송 서비스를 제공하여 전송된 데이터가 분실(lost)되거나 뒤바뀐 순서(out-of-order)로 수신될 수 있음

> **"애플리케이션 관점"**
> UDP는 클라이언트-서버 간 데이터그램을 비신뢰적으로 전달함 (unreliable transfer of datagrams)

리눅스 소켓 프로그래밍 in C

- UDP 절차



리눅스 소켓 프로그래밍 in C

■ 소켓 관련 시스템 콜

```
#include <sys/socket.h>
int socket(int domain, int type, int protocol)
<return 값> 성공: 소켓 디스크립터, 실패: -1
```

- domain, type, protocol에 따라 소켓 생성
- domain: 프로토콜 패밀리를 나타냄
  - ✓ PF_INET 또는 AF_INET: IPv4
  - ✓ PF_INET6 또는 AF_INET6: IPv6
- type: 전송 서비스의 종류를 지정
  - ✓ SOCK_STREAM: 스트림 전송 서비스 (TCP)
  - ✓ SOCK_DGRAM: 데이터그램 전송 서비스 (UDP)
- protocol: 전송 프로토콜
  - ✓ IPPROTO_TCP: TCP 프로토콜
  - ✓ IPPROTO_UDP: UDP 프로토콜

리눅스 소켓 프로그래밍 in C

UDP 소켓 관련 시스템 콜: sendto()

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto(int sockfd, char *buf, int length, int flags,
           struct sockaddr *dest_addr, socklen_t dest_addr_len)
<return 값> 성공: 송신 데이터의 바이트 수, 실패: -1
```

- sockfd가 가리키는 소켓을 통해 buf에 저장된 데이터를 length 길이만큼 dest_addr로 전송
- sockfd: 데이터를 전송하고자 하는 소켓 디스크립터
  - ✓ socket()에서 리턴된 값
- buf: 송신 데이터가 저장된 버퍼
- length: 전송할 데이터의 크기
- flags: 일반적으로 0
- dest_addr: 수신자 주소를 나타내는 구조체
- dest_addr_len: 수신자 주소 구조체의 크기

리눅스 소켓 프로그래밍 in C

UDP 소켓 관련 시스템 콜: recvfrom()

```
#include <sys/types.h>
#include <sys/socket.h>
int recvfrom(int sockfd, char *buf, int length, int flags,
             struct sockaddr *from_addr, socklen_t *from_addr_len)
<return 값> 성공: 수신 데이터의 바이트 수, 실패: -1
```

- sockfd가 가리키는 소켓을 통해 from_addr이 보낸 데이터를 length 길이의 데이터 버퍼 buf로 수신
- sockfd: 데이터를 수신하고자 하는 소켓 디스크립터
  - ✓ socket()에서 리턴된 값
- buf: 수신 데이터가 저장될 버퍼
- length: 수신 데이터 버퍼의 크기
- flags: 일반적으로 0
- from_addr: 송신자 주소를 나타내는 구조체
- from_addr_len: 송신자 주소 구조체의 크기

리눅스 소켓 프로그래밍 in C

bind() 시스템 콜

```
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr *myaddr, socklen_t myaddr_len)
<return 값> 성공: 0, 실패: -1
```

- 소켓을 특정 주소(IP주소와 포트번호)로 바인딩
- sockfd: 바인딩하고자 하는 소켓
- myaddr: 소켓에 바인딩하고자 하는 주소(IP 주소와 포트번호)
- myaddr_len: 바인딩하는 주소 구조체의 크기

```
struct sockaddr {
    sa_family_t sin_family;
    unsigned char sa_data[14];
}
```

```
struct sockaddr_in {
    sa_family_t sin_family;
    unsigned short int sin_port
    struct in_addr sin_addr;
    unsigned char sin_zero[8];
}
struct in_addr {
    unsigned long s_addr;
}
```

리눅스 소켓 프로그래밍 in C

■ 기타 중요 함수

```
#include <netinet/in.h>
uint16_t htons(uint16_t hostshort);
uint16_t ntohs(uint16_t netshort);
```

- 16비트 hostshort/netshort을 네트워크 바이트 순서/호스트 바이트 순서로 바꿔주는 함수

```
#include <netinet/in.h>
uint32_t htonl(uint32_t hostlong);
uint32_t ntohl(uint32_t netlong);
```

- 32비트 hostlong/netlong을 네트워크 바이트 순서/호스트 바이트 순서로 바꿔주는 함수

```
#include <arpa/inet.h>
unsigned long inet_addr(const char *ptr);
char *inet_ntoa(struct in_addr addr);
```

- inet_addr(): 점 10진법 IP주소(문자열) → 이진 IP주소(정수, 네트워크 바이트 순서) 변환
  - ✓ "10.0.0.1" → 0x0a000001
- inet_ntoa(): 이진 IP주소(정수, 네트워크 바이트 순서) → 점 10진법 IP주소(문자열) 변환

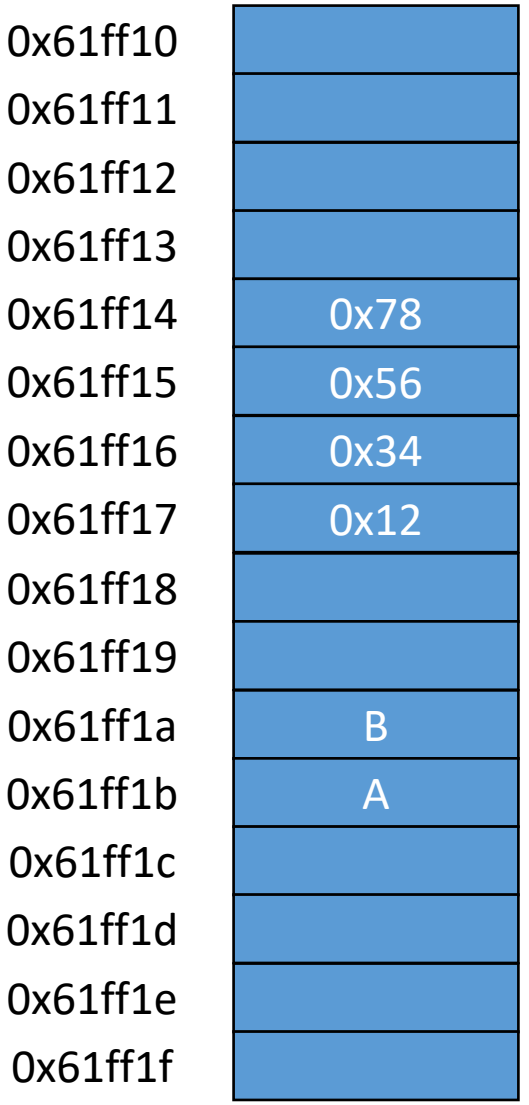
엔디언

■ 메모리 저장 예제

```
#include <stdio.h>

int main()
{
    char a = 'A';
    char b = 'B';
    int c = 0x12345678;
    char *p = &c;
    printf("%c is stored at %x\n", a, &a);
    printf("%c is stored at %x\n", b, &b);
    printf("Addr(%x %x %x %x) Value(%x %x %x %x)\n", \
    p, p+1, p+2, p+3, *p, *(p+1), *(p+2), *(p+3));
}
```

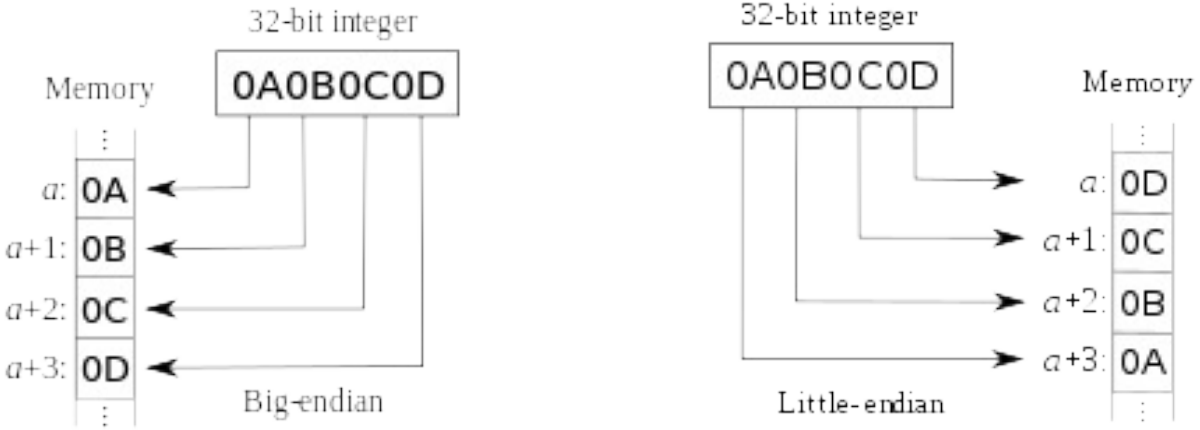
```
A is stored at 61ff1b
B is stored at 61ff1a
Addr(61ff14 61ff15 61ff16 61ff17) Value(78 56 34 12)
```



엔디언

엔디언endian

- 컴퓨터 메모리에 여러 개의 연속된 대상을 배열하는 방법
- 즉, 여러 바이트로 구성된 데이터를 메모리에 저장할 때 어떤 바이트 순서로 저장할 것인지는 나타냄
- 빅 엔디언big endian
  - ✓ 사람이 숫자를 쓰는 방법과 같이 큰 단위의 바이트가 앞에 오는 방법
  - ✓ 모토로라 CPU 계열, 또는 네트워크 전송 시 사용
- 리틀 엔디언little endian
  - ✓ 작은 단위의 바이트가 앞에 오는 방법
  - ✓ 인텔 CPU



엔디언 정리

- 각 시스템은 CPU에 따라 고유의 엔디언 구조를 가짐
  - 이를 "호스트 바이트 순서"라고 함
  - 예) 인텔 CPU: 리틀 엔디언, 모토로라 CPU: 빅 엔디언

- 서로 다른 엔디언 구조를 가진 시스템 간 네트워크 통신을 하기 위해서는 통일된 바이트 순서가 필요함
  - 이를 "네트워크 바이트 순서"라고 하고, "빅 엔디언" 방식을 사용함

- 소켓 프로그래밍 시 유의사항
  - 네트워크로 데이터 전송 시 "네트워크 바이트 순서"로 변환하여 전송해야 함
  - 네트워크에서 데이터 수신 시 "호스트 바이트 순서"로 변환하여 처리해야 함

리눅스 소켓 프로그래밍 in C UDP 서버

udp_server.c

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

#define BUFFER_SIZE  1024

int main(int argc, char *argv[]) {
    int sock;                           // UDP 서버 쪽 소켓 디스크립터
    struct sockaddr_in srvAddr;         // 서버 주소
    struct sockaddr_in cliAddr;         // 클라이언트 주소
    unsigned short srvPort;
    char buffer[BUFFER_SIZE];
    int sentSize, rcvSize;
    unsigned int srvAddrLen, cliAddrLen;
    int ret, i;

    if (argc != 2) {
        printf("Usage: %s Port\n", argv[0]);
        exit(0);
    }
    srvPort = atoi(argv[1]);            // 서버 포트를 명령 실행줄에서 입력 받음
```

리눅스 소켓 프로그래밍 in C UDP 서버

```
// UDP 용 소켓 생성
// 이후 이 소켓을 사용할 때에는 "sock" 소켓 디스크립터를 사용
sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (sock < 0) {
    printf("socket() failed\n");
    exit(0);
}

// 서버 주소 설정. 이후 바인드 시스템 콜에서 사용됨
memset(&srvAddr, 0, sizeof(srvAddr));                    // 주소를 0으로 초기화
srvAddr.sin_family = AF_INET;                            // IPv4
srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);             // 서버 IP는 지정하지 않음
srvAddr.sin_port = htons(srvPort);                       // 서버 포트번호 지정

// 소켓과 서버 주소를 바인딩
ret = bind(sock, (struct sockaddr *)&srvAddr, sizeof(srvAddr));
if (ret < 0) {
    printf("bind() failed\n");
    exit(0);
}

printf("Server is running.\n");
```

리눅스 소켓 프로그래밍 in C UDP 서버

```
    while (1) {     // 무한루프를 돌면서 메시지 송수신
        cliAddrLen = sizeof(cliAddr);
        rcvSize = recvfrom(sock, buffer, BUFFER_SIZE, 0, (struct sockaddr *)&cliAddr,
                  &cliAddrLen);             // sock 소켓을 통해 buffer로 메시지 수신
        if (rcvSize < 0) {
            printf("Error in recvfrom()\n");
            exit(0);
        }

        printf("[Client/%s:%d] %s\n", inet_ntoa(cliAddr.sin_addr),
            ntohs(cliAddr.sin_port), buffer);

        if (!strcmp(buffer, "quit")) break;     // 입력받은 데이터가 quit이면 종료

        for (i = 0; buffer[i] != '\0'; i++)     // 대문자로 변환
            buffer[i] = toupper(buffer[i]);

        sentSize = sendto(sock, buffer, strlen(buffer)+1, 0, (struct sockaddr *)&cliAddr,
                       sizeof(cliAddr));        // sock 소켓을 통해 변환한 데이터 전송
        if (sentSize != strlen(buffer)+1) {
            printf("sendto() sent a different number of bytes than expected");
            exit(0);
        }
    }
    close(sock);
    printf("UDP Server is Closed.\n");
}
```

리눅스 소켓 프로그래밍 in C UDP 클라이언트

udp_client.c

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

#define BUFFER_SIZE    1024

int main(int argc, char *argv[]) {
    int sock;                           // UDP 클라이언트 쪽 소켓 디스크립터
    struct sockaddr_in srvAddr;         // 서버 주소
    char *srvIp = NULL;
    unsigned short srvPort;
    char buffer[BUFFER_SIZE];
    int sentSize, rcvSize;
    unsigned int srvAddrLen;

    if (argc != 3) {
        printf("Usage: %s IP_addr Port\n", argv[0]);
        exit(0);
    }
    srvIp = argv[1];                    // 서버 IP를 명령 실행줄에서 입력 받음
    srvPort = atoi(argv[2]);            // 서버 포트를 명령 실행줄에서 입력 받음
```

리눅스 소켓 프로그래밍 in C UDP 클라이언트

```
// UDP 용 소켓 생성
// 이후 이 소켓을 사용할 때에는 "sock" 소켓 디스크립터를 사용
sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (sock < 0) {
    printf("socket() failed\n");
    exit(0);
}

// 서버 주소 설정. 이후 서버로 데이터 전송 시 사용됨
memset(&srvAddr, 0, sizeof(srvAddr));          // 주소를 0으로 초기화
srvAddr.sin_family = AF_INET;                  // IPv4
srvAddr.sin_addr.s_addr = inet_addr(srvIp);    // 서버 IP 지정
srvAddr.sin_port = htons(srvPort);             // 서버 포트 번호 지정

printf("Client is running.\n");
printf("Enter the word to translate into capitals\n");

while (1) {      // 무한루프를 돌면서 메시지 송수신
    fgets(buffer, BUFFER_SIZE, stdin);         // 키보드에서 buffer로
    buffer[strlen(buffer)-1] = '\0';           // 한 줄씩 입력 받음
                                               // 마지막에 널 문자 추가
```

리눅스 소켓 프로그래밍 in C UDP 클라이언트

```
        sentSize = sendto(sock, buffer, strlen(buffer)+1, 0,
                        (struct sockaddr *)&srvAddr, sizeof(srvAddr));
        if (sentSize != strlen(buffer)+1) {
            printf("sendto() sent a different number of bytes than expected\n");
            exit(0);
        }

        if (!strcmp(buffer, "quit")) break;

        srvAddrLen = sizeof(srvAddr);
        rcvSize = recvfrom(sock, buffer, BUFFER_SIZE, 0,
                        (struct sockaddr *)&srvAddr, &srvAddrLen);
        if (rcvSize < 0) {
            printf("Error in recvfrom()\n");
            exit(0);
        }

        printf("[Server/%s:%d] %s\n", inet_ntoa(srvAddr.sin_addr),
            ntohs(srvAddr.sin_port), buffer);     // 수신한 데이터를 화면에 출력
    }
    close(sock);
    printf("UDP Client is Closed.\n");
}
```

실행 화면

- UDP 서버 (항상 서버부터 실행)
  - 자신의 포트 번호만 지정함 (IP는 자동 지정)

```
root@kali:~# ./udp_server 7777
Server is running.
[Client/127.0.0.1:40006] hello
[Client/127.0.0.1:40006] IoT
[Client/127.0.0.1:40006] My name is Daehee Kim.
[Client/127.0.0.1:40006] My ID is 20170031.
[Client/127.0.0.1:40006] Bye!
[Client/127.0.0.1:40006] quit
UDP Server is Closed.
root@kali:~#
```

- UDP 클라이언트
  - 서버의 IP 주소와 포트 번호를 지정함 (자신의 IP와 포트번호는 자동 지정)

```
root@kali:~# ./udp_client 127.0.0.1 7777
Client is running.
Enter the word to translate into capitals
hello
[Server/127.0.0.1:7777] HELLO
IoT
[Server/127.0.0.1:7777] IOT
My name is Daehee Kim.
[Server/127.0.0.1:7777] MY NAME IS DAEHEE KIM.
My ID is 20170031.
[Server/127.0.0.1:7777] MY ID IS 20170031.
Bye!
[Server/127.0.0.1:7777] BYE!
quit
UDP Client is Closed
root@kali:~#
```

TCP 소켓 프로그래밍

- 클라이언트는 서버에 접속해야 함
  - 서버 프로세스가 먼저 수행 중에 있어야 함
  - 서버는 클라이언트의 초기 접속을 처리하는 소켓을 생성해야 함
  - 클라이언트는 TCP 소켓을 생성하고, 서버 프로세스의 IP 주소와 포트 번호를 명시한 후, 서버에 접속하여 TCP 연결 설정

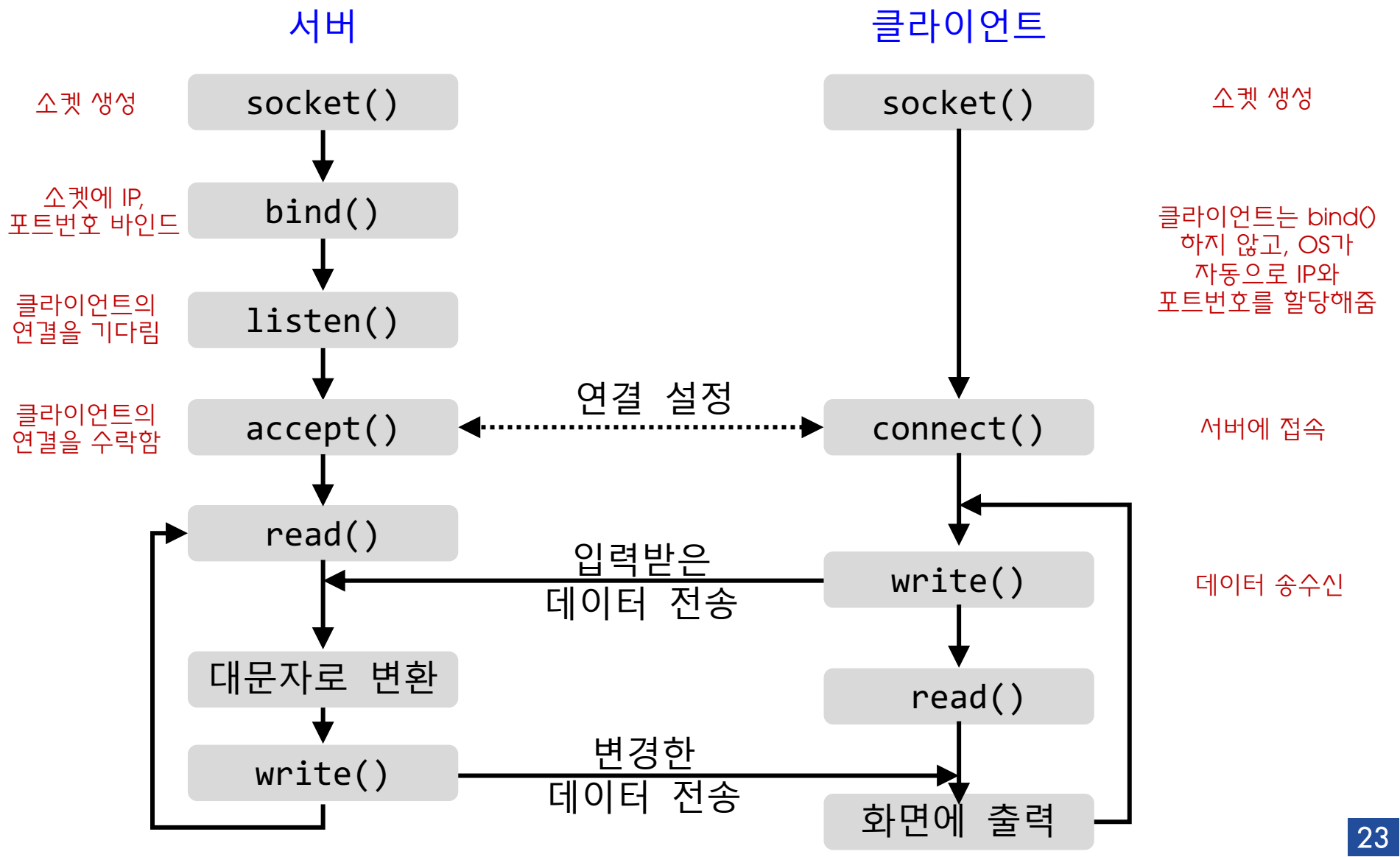
- 클라이언트 접속 시, 서버는 클라이언트와 통신하는 새로운 소켓(연결 소켓)을 생성
  - 서버가 다수의 클라이언트와 통신할 수 있도록 해줌
  - 소스 포트 번호가 클라이언트들을 구분함

애플리케이션 관점

TCP는 클라이언트-서버 간 신뢰성 있고, 순서가 보장된 바이트 스트림 전송을 제공

리눅스 소켓 프로그래밍 in C

- TCP 절차



리눅스 소켓 프로그래밍 in C

TCP 소켓 관련 시스템 콜: listen(), connect()

```
#include <sys/socket.h>
int listen(int sockfd, int backlog)
<return 값> 성공: 0, 실패: -1
```

- sockfd가 가리키는 소켓을 통해 클라이언트의 연결을 기다리는 시스템 콜. TCP 서버 쪽에서 사용됨
- sockfd: 연결을 기다리는 소켓 디스크립터
- backlog: 동시에 서비스 가능한 요청의 개수

```
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr *srv_addr,
            socklen_t srv_addr_len)
<return 값> 성공: 0, 실패: -1
```

- 클라이언트가 서버에 TCP 연결을 요청하는 시스템 콜
- sockfd: 연결 요청을 보낼 소켓 디스크립터
- srv_addr: 서버 주소를 나타내는 구조체
- srv_addr_len: 서버 주소 구조체의 크기

리눅스 소켓 프로그래밍 in C

TCP 소켓 관련 시스템 콜: accept()

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *cli_addr,
           socklen_t * cli_addr_len)
<return 값> 성공: 새로운 소켓 디스크립터, 실패: -1
```

- 클라이언트의 연결 요청을 허락하는 시스템 콜
- 클라이언트와 TCP 통신을 하기 위해 새로운 소켓을 생성하여 리턴함
- sockfd: 연결을 기다리는 소켓 디스크립터
- cli_addr: 송신자(클라이언트) 주소를 나타내는 구조체
- cli_addr_len: 송신자(클라이언트) 주소 구조체의 크기

리눅스 소켓 프로그래밍 in C

TCP 소켓 관련 시스템 콜: read(), write()

```
#include <unistd.h>
int read(int sockfd, char *buf, int nbytes)
<return 값> 성공: 수신한 바이트 수, 실패: -1
```

- sockfd가 가리키는 소켓을 통해 최대 nbytes 만큼 수신하는 시스템 콜
- sockfd: 데이터를 수신할 소켓 디스크립터
- buf: 수신한 데이터를 저장할 버퍼
- nbytes: 최대 수신 바이트 수

```
#include <unistd.h>
int write(int sockfd, char *buf, int nbytes)
<return 값> 성공: 송신한 바이트 수, 실패: -1
```

- sockfd가 가리키는 소켓을 통해 nbytes 만큼 송신하는 시스템 콜
- sockfd: 데이터를 송신할 소켓 디스크립터
- buf: 송신할 데이터가 저장된 버퍼
- nbytes: 송신 바이트 수

리눅스 소켓 프로그래밍 in C TCP 서버

tcp_server.c

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

#define BUFFER_SIZE  1024

int main(int argc, char *argv[]) {
    int sSock, listener;
    struct sockaddr_in srvAddr, cliAddr;
    unsigned short srvPort;
    char buffer[BUFFER_SIZE];
    int sentSize, rcvSize;
    unsignd int srvAddrLen, cliAddrLen;
    int ret, i;

    if (argc != 2) {
        printf("Usage: %s Port\n", argv[0]);
        exit(0);
    }
    srvPort = atoi(argv[1]);          // 서버 포트를 명령 실행줄에서 입력 받음
```

리눅스 소켓 프로그래밍 in C TCP 서버

```
// TCP 서버 용 소켓 생성. 클라이언트로부터 요청을 받기 위해 사용되는 소켓
// 클라이언트와 데이터 송수신 시에는 다른 소켓이 사용됨
listener = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (listener < 0) {
    printf("Server socket() failed\n");
    exit(0);
}

// 서버 주소 설정. 이후 바인드 시스템 콜에서 사용됨
memset(&srvAddr, 0, sizeof(srvAddr));              // 주소를 0으로 초기화
srvAddr.sin_family = AF_INET;                      // IPv4
srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);       // 서버 IP는 지정하지 않음
srvAddr.sin_port = htons(srvPort);                 // 서버 포트번호 지정

// 소켓과 서버 주소를 바인딩
ret = bind(listener, (struct sockaddr *)&srvAddr, sizeof(srvAddr));
if (ret < 0) {
    printf("Server cannot bind local addresss.\n");
    exit(0);
}

printf("Server is running.\n");
```

리눅스 소켓 프로그래밍 in C TCP 서버

```
//클라이언트의 연결을 기다림. 최대 5개의 클라이언트 동시 서비스 가능
ret = listen(listener, 5);
if (ret < 0) {
    printf("Server failed to listen.\n");
    exit(0);
}

cliAddrLen = sizeof(cliAddr);
// 클라이언트 연결 수락. 클라이언트와 통신할 새로운 소켓(sSock) 생성
sSock = accept(listener, (struct sockaddr *)&cliAddr, &cliAddrLen);
if (sSock < 0) {
    printf("Server failed to accept.\n");
    exit(0);
}
printf("Client is connected.\n");

while (1) {        // 무한루프를 돌면서 메시지 송수신
    rcvSize = read(sSock, buffer, BUFFER_SIZE);     // 메시지 수신
    if (rcvSize < 0) {
        printf("Error in read()\n");
        exit(0);
    }

    printf("[Client/%s:%d] %s\n", inet_ntoa(cliAddr.sin_addr),
        ntohs(cliAddr.sin_port), buffer);

    if (!strcmp(buffer, "quit")) break;
```

리눅스 소켓 프로그래밍 in C TCP 서버

```
        for (i = 0; buffer[i] != '\0'; i++)
            buffer[i] = toupper(buffer[i]);

        sentSize = write(sSock, buffer, strlen(buffer)+1);      // 메시지 송신
        if (sentSize != strlen(buffer)+1) {
            printf("write() sent a different number of bytes than expected\n");
            exit(0);
        }
    }

    close(sSock);
    close(listener);
    printf("TCP Server is Closed.\n");
}
```

리눅스 소켓 프로그래밍 in C TCP 클라이언트

tcp_client.c

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

#define BUFFER_SIZE  1024

int main(int argc, char *argv[]) {
    int cSock;
    struct sockaddr_in srvAddr;
    char *srvIp = NULL;
    unsigned short srvPort;
    char buffer[BUFFER_SIZE];
    int sentSize, rcvSize;
    int ret;

    if (argc != 3) {
        printf("Usage: %s IP_addr Port\n", argv[0]);
        exit(0);
    }
    srvIp = argv[1];                // 서버 IP를 명령 실행줄에서 입력 받음
    srvPort = atoi(argv[2]);        // 서버 포트를 명령 실행줄에서 입력 받음
```

리눅스 소켓 프로그래밍 in C TCP 클라이언트

```
// TCP 클라이언트 용 소켓 생성
cSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (cSock < 0) {
    printf("socket() failed\n");
    exit(0);
}

// 서버 주소 설정. 이후 서버로 접속 시 connect() 시스템 콜에서 사용됨
memset(&srvAddr, 0, sizeof(srvAddr));                // 주소를 0으로 초기화
srvAddr.sin_family = AF_INET;                        // IPv4
srvAddr.sin_addr.s_addr = inet_addr(srvIp);          // 서버 IP 지정
srvAddr.sin_port = htons(srvPort);                   // 서버 포트 번호 지정

// 서버에 TCP 연결 수행
ret = connect(cSock, (struct sockaddr *)&srvAddr, sizeof(srvAddr));
if (ret < 0) {
    printf("Client cannot connect to the Server.\n");
    exit(0);
}
printf("Client is running.\n");
printf("Enter the word to translate into capitals\n");

while (1) {       // 무한루프를 돌면서 메시지 송수신
    fgets(buffer, BUFFER_SIZE, stdin);
    buffer[strlen(buffer)-1] = '\0';
```

리눅스 소켓 프로그래밍 in C TCP 클라이언트

```
        sentSize = write(cSock, buffer, strlen(buffer)+1);
        if (sentSize != strlen(buffer)+1) {
            printf("write() sent a different number of bytes than expected\n");
            exit(0);
        }

        if (!strcmp(buffer, "quit")) break;

        rcvSize = read(cSock, buffer, BUFFER_SIZE);
        if (rcvSize < 0) {
            printf("Error in read()\n");
            exit(0);
        }

        printf("[Server/%s:%d] %s\n", inet_ntoa(srvAddr.sin_addr),
            ntohs(srvAddr.sin_port), buffer);
    }

    close(cSock);
    printf("TCP Client is Closed.\n");
}
```

실행 화면

- TCP 서버 (항상 서버부터 실행)
  - 자신의 포트 번호만 지정함 (IP는 자동 지정)

```
root@kali: ~# ./tcp_server 1234
Server is running.
[Client/127.0.0.1:51294] hello
[Client/127.0.0.1:51294] IoT
[Client/127.0.0.1:51294] My name is Daehee Kim.
[Client/127.0.0.1:51294] My ID is 20170031.
[Client/127.0.0.1:51294] Bye!
[Client/127.0.0.1:51294] quit
TCP Server is Closed.
root@kali: ~#
```

- TCP 클라이언트
  - 서버의 IP 주소와 포트 번호를 지정함 (자신의 IP와 포트번호는 자동 지정)

```
root@kali: ~# ./tcp_client 127.0.0.1 1234
Client is running.
Enter the word to translate into capitals
hello
[Server/127.0.0.1:1234] HELLO
IoT
[Server/127.0.0.1:1234] IOT
My name is Daehee Kim.
[Server/127.0.0.1:1234] MY NAME IS DAEHEE KIM.
My ID is 20170031
[Server/127.0.0.1:1234] MY ID IS 20170031.
Bye!
[Server/127.0.0.1:1234] BYE!
quit
TCP Client is Closed
root@kali: ~#
```

프로그래밍 과제 1

- 본 슬라이드의 소켓 프로그래밍 직접 해보기
  - 리눅스(in C)에서 UDP 소켓 프로그래밍 (C 서버/ C 클라이언트 간)
  - 리눅스(in C)에서 TCP 소켓 프로그래밍 (C 서버/ C 클라이언트 간)

  - 제출기한: 과제 공지 후 1주
    - ✓ LMS의 "(프로그래밍_과제_1)소켓_프로그래밍"에 소스코드 및 실행 캡쳐화면 제출
    - ✓ 제출파일
      - 각 case에 대한 소스코드(서버, 클라이언트 코드)
      - 각 case에 대한 실행 캡쳐화면(서버, 클라이언트 실행 화면)

프로그래밍 과제 2

■ 채팅 프로그램 만들기

- 클라이언트와 서버 간 UDP로 메시지를 주고 받는 채팅 프로그램 작성
- 동작
  - ✓ 클라이언트와 서버가 번갈아 가면서 키보드에서 1문장씩 입력 받음 (무한 루프)
    - "클라이언트"부터 시작
  - ✓ 1문장이 입력되면 상대방에게로 해당 메시지를 전달하고, 화면에 출력
    - 출력 포맷은 "메시지"
    - 본인의 IP 주소와 포트 번호는 출력할 필요 없음
  - ✓ 1문장을 상대방으로부터 수신하면, 화면에 출력
    - 출력 포맷은 "[상대방 IP주소:포트 번호] 메시지"
  - ✓ 아무 쪽에서나 "quit"을 입력하면 채팅 종료
    - 클라이언트/서버 모두 "Chat Closed" 출력 후 소켓 닫고 프로그램 종료
- 주의사항
  - ✓ 채팅 메시지는 "영어"로 진행
- 제출기한: 과제 공지 후 2주
  - ✓ LMS의 "(프로그래밍_과제_2)_간단한_채팅_프로그램"에 소스코드 및 실행 캡쳐화면 제출
  - ✓ 제출파일: 소스코드(서버, 클라이언트 코드), 실행 캡쳐화면(서버, 클라이언트 실행 화면)

프로그래밍 과제 3

```
HTTP/1.1 200 OK
Date: Thu, 03 Mar 2022 11:34:57 GMT
Server: Apache
Set-Cookie: elevisor_for_j2ee_uid=554y9drkvr8y4; Expires=Fri, 03-Mar-2023 11:34:57 GMT; Path=/
Set-Cookie: JSESSIONID=Hp9P2gi29Ng0jK9Lm+p2W8nt.mehome2; Path=/
Content-Length: 602
Content-Type: text/html;charset=UTF-8

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ko" lang="ko">
<head>
<title>?쌜꼍遺□?댔럹?댁?</title>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta charset="utf-8">
<meta name="description" content="">
<meta name="format-detection" content="telephone=no">
<meta name="viewport" content="width=device-width, height=device-height">
<script type="text/javascript" src="/jquery/jquery-3.1.1.js"></script>
<script type="text/javascript">
//<![CDATA[
location.href = "http://me.go.kr/home/web/main.do";
//]]>
</script>
</head>
<body>
</body>
</html>
```

■ 프로그래밍 과제 3

- 간단한 웹 클라이언트 프로그램 작성
- 동작
  - ✓ 접속하고자 하는 웹 서버로 TCP 연결
    - 웹 서버의 IP 주소는 명령행에서 입력
    - 포트 번호는 기본으로 80번 사용
  - ✓ 웹 서버로 아래 문자열을 차례로 전송 (HTTP Request)
    - `"GET / HTTP/1.1\r\n"`
    - `"Host: me.go.kr\r\n"`
    - `"\r\n"`
  - ✓ 웹 서버로부터 받은 응답을 문자열로 출력 (HTTP Response)
- 프로그램 실행 방법
  - ✓ [실행파일 이름] 27.101.216.200
    - 예)./webclient 27.101.216.200
- 제출기한: 과제 공지 후 3주
  - ✓ LMS의 "(프로그래밍_과제_3)_간단한_웹_클라이언트"에 소스코드 및 실행 캡쳐화면 제출
  - ✓ 제출파일: 소스코드(웹 클라이언트 코드), 실행 캡쳐화면(웹 클라이언트 실행 화면 앞부분)

결과 출력시, 한글은 깨져도 상관없으며, 내용이 위와 다를 수 있음

27.101.216.200은 me.go.kr의 IP 주소임