

네트워킹-1

AMM42020 게임서버 프로그래밍
정내훈

2023년도 1학기

한국공학대학교 게임공학과

목차

- 네트워크 복습
- Socket Programming 복습

네트워크

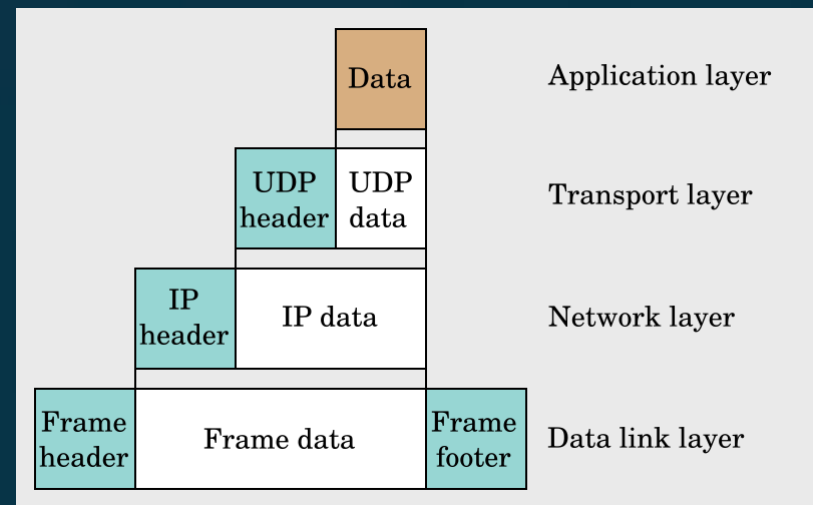
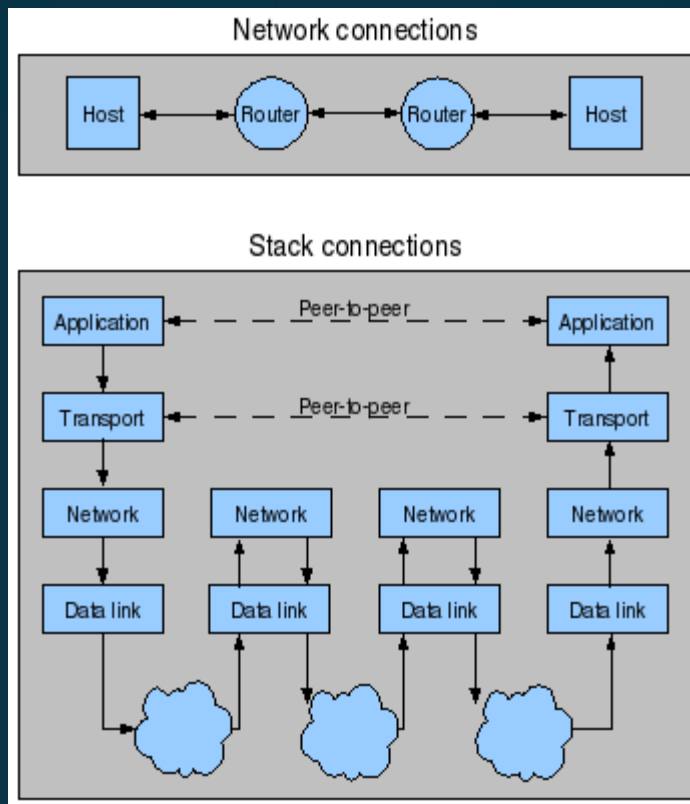
- 네트워크 기초에서 배운 내용
 - 용어
 - 프로토콜, 패킷, 인터넷 레이어
 - 인터넷의 구조
 - Routing, Domain Name, TCP/UDP, IPv4/IPv6

복습

- 인터넷은 4개의 Layer로 구현된다.
 - Application : 우리가 작성하는 프로그램, 게임 클라이언트, **게임 서버**
 - Transport : Application에 데이터를 전달해 주는 API, TCP, UDP (운영체제 레벨에서 구현)
 - Network : 데이터를 목적지 까지 안내해 주는 시스템, IPv4/IPv6
 - Data Link : 컴퓨터와 컴퓨터를 연결하는 물리적인 규격 (Ethernet, PPP, FDDI, LTE/5G, WIFI)

복습

● 데이터의 흐름



Ref: <http://en.wikipedia.org/wiki/TCP/IP>

패킷

- 패킷

- 인터넷 표준에서의 패킷

- 레이어에서 다루는 데이터의 단위
 - 레이어 마다 한번에 다룰 수 있는 데이터의 최대 크기가 있음
 - 각 레이어에서 필요한 정보가 추가됨 (헤더)
 - 패킷은 독립적이어야 함, 앞/뒤의 데이터를 보지 않아도 전달할 곳을 알 수 있어야 함.

- Application에서의 패킷

- 구분을 위해 다른 이름으로 부르는 사람도 많음
 - 구현의 편의성을 위해 프로그래머가 나누어 놓은 데이터의 단위
 - 패킷은 독립적이어야 함, 앞/뒤의 데이터를 보지 않아도 무슨 내용인지 알 수 있어야 함.

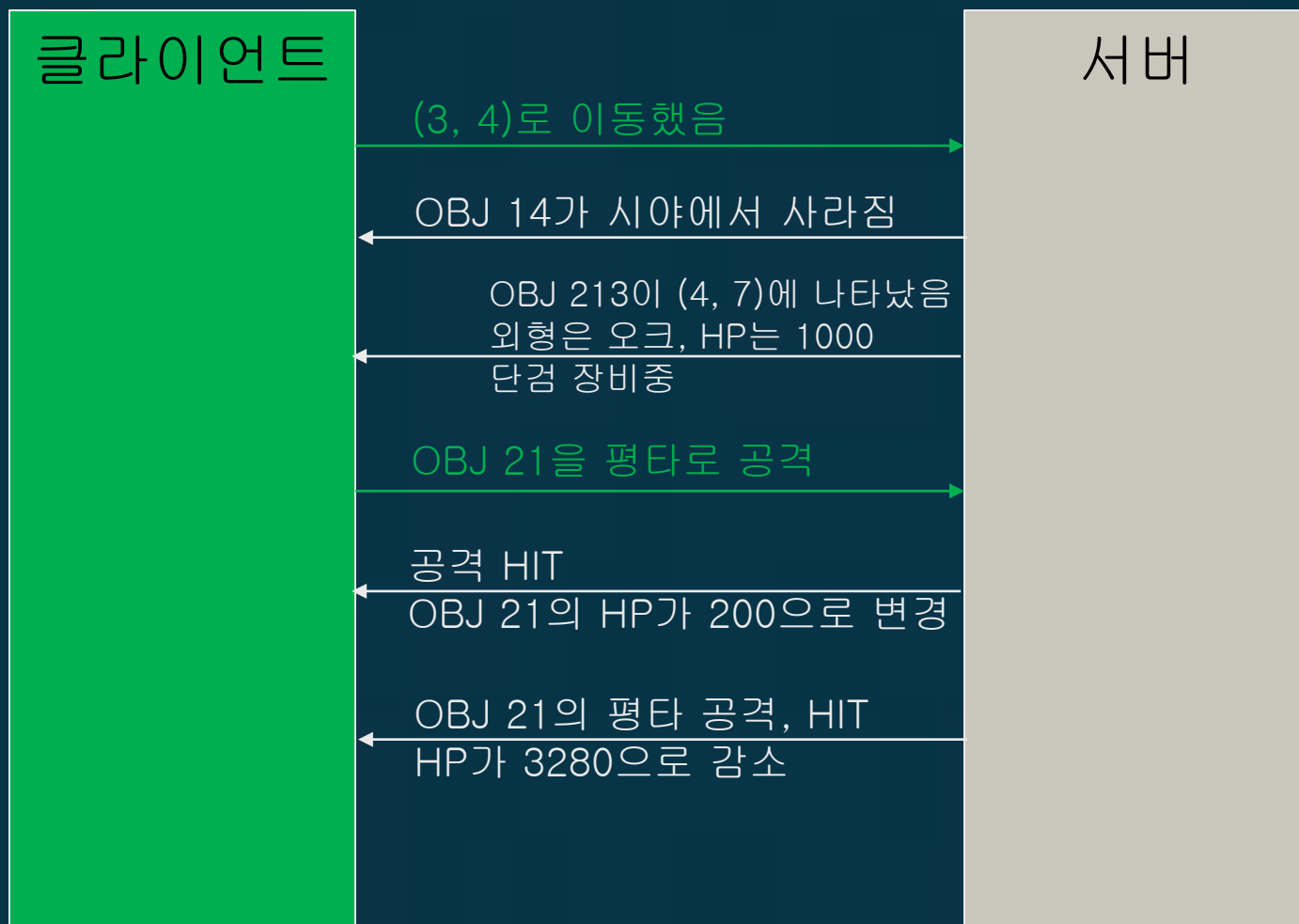
패킷

● MMO 패킷의 특징

- 종류가 많다
 - 이동, 아이템사용, 평타, 채팅, 몬스터 공격, 경매, 파티...
- 다음에 어떤 패킷이 올지 알 수 없다.
 - 모든 패킷을 한번씩 순서대로 보낼 수 없다.
- 패킷마다 크기가 다 다르다.
 - 제일 큰 패킷의 크기로 통일하면 => 네트워크 낭비
- 따라서, 패킷을 받았을 때 패킷의 종류와 크기를 알 수 있어야 한다.
 - 패킷 맨 앞에 크기와, 종류를 먼저 적어야 한다!!

패킷

- MMO에서의 패킷



패킷

- 프로토콜

- 클라이언트와 서버가 주고 받는 데이터의 **포맷**과 **순서**를 정의
- **패킷 포맷** 정의가 핵심
- MMO에서는 수많은 종류의 **Packet**이 존재
 - 로그인, 캐릭터 선택, 이동, 아이템 사용, 채팅, 평타, 마법, 스킬 사용, 데미지, 상태 변경...
- 주의점 : 수정은 클라이언트와 서버가 동시에 이루어져야 한다.

복습

- 프로토콜 정의 방식
 - Binary Format
 - Byte단위로 값의 의미를 정의
 - Structure Format
 - C의 구조체를 정의해서 공유
 - SDK 사용
 - 프로토콜의 정의와 해석을 도와주는 유틸리티
 - OpenSource : Protocol Buffer, Flat Buffer
 - 웹에서는 XML이나 Javascript도 많이 사용.

복습

● 프로토콜 : Binary Format

- 관리가 까다로움, 언어/운영체제/컴파일러에 구애 받지 않음, Endian만 주의하면 됨.

For move

size	type	x	y	z	dx	dy	dz
0	2	3	7	11	15	19	23

```
char buf[256];
```

```
*((short *)(&buf[0])) = size;    buf[2] = OP_MOVE;
*((float *)(&buf[3])) = x;        *((float *)(&buf[7])) = y;
*((float *)(&buf[11])) = z;       *((float *)(&buf[15])) = dx;
*((float *)(&buf[19])) = dy;      *((float *)(&buf[23])) = dz;
```

```
...
```

```
send( fd, buf, ((short *)buf)[0], 0 );
```

복습

- 프로토콜 : Structure Format

- 관리가 쉬움, 언어/운영체제/컴파일러에 구애받는다.

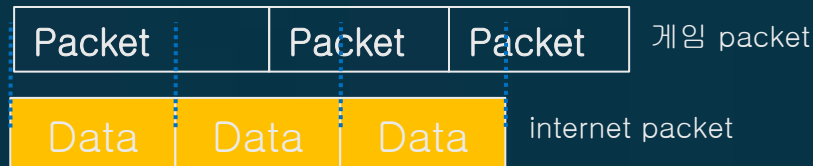
```
#pragma pack (push, 1)
struct move_packet {
    short size;
    char  type
    float x, y, z;
    float dx, dy, dz;
}
#pragma pack (pop)
```

```
move_packet p;
p.size = sizeof(p);
p.type = OP_MOVE
p.x = obj->x;
p.y = obj->y;
p.z = obj->z;
p.dx = obj->dx;
p.dy = obj->dy;
p.dz = obj->dz;
...
send(socket, &p, p.size, 0);
```

패킷

● 패킷 재조립

- (인터넷 표준에서의 패킷) ≠ (Application에서의 패킷)
- 100바이트 + 100바이트 + 100바이트를 보냈을 때
상대쪽에서 어떠한 조합으로 도착할 지 알 수 없다.
 - (O) 100바이트 + 100바이트 + 100바이트
 - (O) 10바이트 + 150바이트 + 140바이트
 - (O) 300바이트
 - (O) 20바이트 + 30바이트 + 50바이트 + 40바이트 + 100바이트 + 60바이트
 - (X) 120바이트 + 120바이트 + 120바이트



- 받는 쪽에서의 패킷 재 조립이 필요하다.

목차

- 네트워크 기초 (복습)
- **Socket Programming** 복습

기본 프로그래밍

- 기본은 두개의 프로그램이 서로 데이터를 주고 받는 것.
 - 두개의 프로그램은 인터넷으로 연결된다.
 - 하나는 클라이언트, 하나는 서버
 - 클라이언트는 서버의 주소(위치)를 알아야 한다.
- File I/O와 거의 같다.
 - Open => (클라이언트는 Connect, 서버는 Accept)
 - Read => Recv
 - Write => Send
 - Close => Close 또는 CloseSocket

기본 프로그래밍

- 네트워크 프로그래밍 단계

- 클라이언트

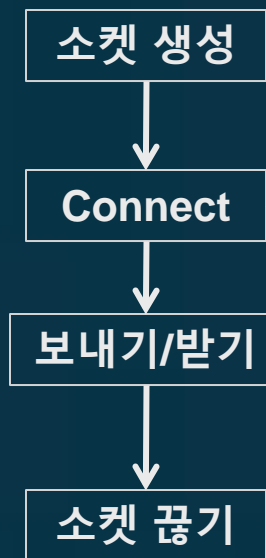
- 소켓 생성 (Socket)
 - 서버 소켓 연결 (Connect)
 - Data 송/수신 (Recv/Send)
 - 소켓 끝기 (Close)

- 서버

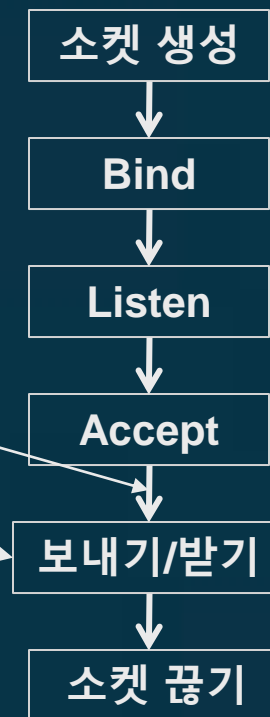
- 소켓 생성 (Socket)
 - 소켓 묶기 (Bind)
 - 소켓 접속 대기 (Listen)
 - 연결 소켓 생성 (Accept)
 - Data 송/수신 (Recv/Send)
 - 소켓 끝기 (Close)

기본 프로그래밍

클라이언트



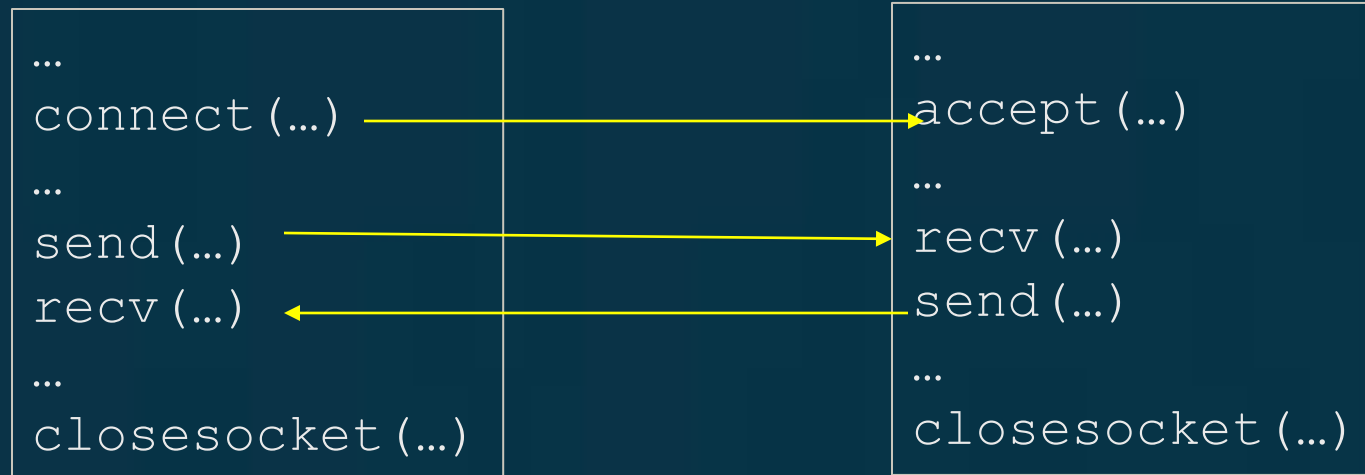
서버



기본 프로그래밍

클라이언트

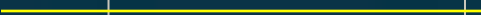
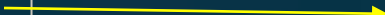
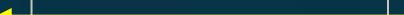
서버



기본 프로그래밍

클라이언트

서버

```
s = socket(...)  
connect(s, ...)   
...  
send(s, buf, ...)   
recv(s, buf, ...)   
...  
closesocket(s)
```

```
s = socket(...)  
c = accept(s)  
...  
recv(c, buf, ...)   
send(c, buf, ...)   
...  
closesocket(c)
```

기본 프로그래밍

클라이언트

```
SOCKET s = socket(AF_INET, SOCK_STREAM, 0);
sockaddr addr;
addr.sin_addr.s_addr = ...;
addr.sin_port = PORT;
connect(s, &addr, ...)
...
sent = send(s, buf, size, flags);
r_size = recv(s, buf, max_size, flags);
...
closesocket(s);
```

서버

```
SOCKET s = socket(AF_INET, SOCK_STREAM, 0);
sockaddr c_addr;
c_addr.sin_port = PORT;
bind(s, &c_addr);
listen(s, 3);
sockaddr n_addr;
SOCKET c = accept(s, &n_addr)
...
r_size = recv(c, buf, max_size, flags);
sent = send(c, buf, size, &sent, flags);
...
closesocket(c);
```

Windows Network

- 실습은 Windows Socket API를 사용
 - 표준 Socket API에 부족한 부분이 많음
 - Windows는 기본 socket API 도 사용 가능
 - socket(), connect(), accept(), send(), recv()
 - 하지만 고성능 네트워크 I/O를 위해서는 전용 API 필요.
 - WSASocket(), WSAConnect(), WSAAccept(), WSASend(), WSARecv()
- 섞어서 사용해도 지금은 문제 없음
 - 뒤에 가면 문제가 생김.

Windows Network (2023)

- Socket 만들기

- SOCKET WSA`Socket`(int af, int type, int protocol, LPWSA`PROTOCOL_INFO` lpProtocolInfo, GROUP g, DWORD dwFlags)
 - af : address family
 - AF_INET만 사용 (AF_NETBIOS, AF_IRDA, AF_INET6)
 - type : 소켓의 타입
 - tcp를 위해 SOCK_STREAM사용 (SOCK_DGRAM)
 - protocol : 사용할 프로토콜 종류
 - IPPROTO_TCP (IPPROTO_UDP)
 - lpProtocolInfo : 프로토콜 정보
 - 보통 NULL
 - g : 예약
 - dwFlags : 소켓의 속성
 - 보통 0 (또는 WSA_PROTOCOL_OVERLAPPED)

Windows Network

- Socket 연결

- 대기하고 있는 상대방소켓에 자신의 소켓을 연결

- `int WSAConnect(SOCKET s, const struct sockaddr* name, int namelen, LPWSABUF lpCallerData, LPWSABUF lpCalleeData, LPQOS lpSQOS, LPQOS lpGQOS)`

- s : 소켓
 - name, namelen : 상대 소켓 주소, 주소 길이
 - 나머지 : 생략

Windows Network

- `sockaddr` 주소 구조체

- 네트워크 주소를 저장

- IP 주소와 PORT 번호

- IP 주소 => 컴퓨터, PORT 번호 => 프로세스

- little endian, big endian 주의

- 선언

- `struct sockaddr addr;`

- 사용법

```
addr.sin_family = AF_INET;  
addr.sin_port = htons(SERVER_PORT);  
inet_pton(AF_INET, SERVER_IP, &addr.sin_addr);
```

- `SERVER_PORT`는 short, `SERVER_IP`는 문자열

Windows Network

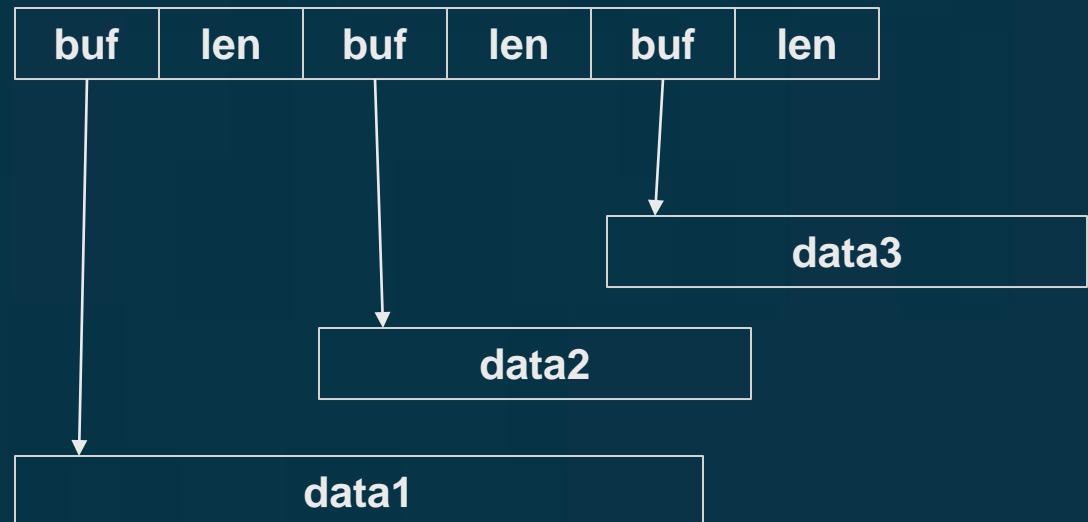
- Socket에서 데이터 받기
 - `int WSARecv(SOCKET s, LPWSABUF lpBuffers, DWORD dwBufferCount, LPDWORD lpNumberOfBytesRecv, LPDWORD lpFlags, LPWSAOVERLAPPED lpOverlapped, LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine)`
 - `s` : 소켓
 - `lpBuffers` : 받은 데이터를 저장할 버퍼
 - `dwBufferCount` : 버퍼의 개수
 - `lpFlags` : 동작 옵션(`MSG_PEEK`, `MSG_OOB`)
 - `lpNumberOfBytesRecv` : 받은 데이터의 크기
 - `lpOverlapped`, `lpCompletionRoutine` : 뒤에 설명

Windows Network

● WSABUF?

- 흠어진 데이터를 관리하는 구조
- 성능 저하의 주범인 데이터 복사를 줄여줌
- **gather/scatter I/O**라고 불림

```
typedef struct __WSABUF
{
    u_long len;
    char FAR *buf;
} WSABUF, *LPWSABUF;
```



Windows Network

- WSABUF를 사용한 최적화
 - Move_p, Attack_p, Chat_p의 3개의 패킷을 보내야 한다면?

```
Send(s, &move_p, sizeof(move_p));  
Send(s, &attack_p, sizeof(attack_p));  
Send(s, &chat_p, sizeof(chat_p));
```

```
Char buf[1024];  
Memcpy(buf, &move_p, sizeof(move_p));  
Memcpy(buf + sizeof(move_p), &attack_p, sizeof(attack_p));  
Memcpy(buf + sizeof(move_p) + sizeof(attack_p), &chat_p, sizeof(chat_p));  
Send(s, buf, sizeof(move_p) + sizeof(attack_p) + sizeof(chat_p));
```

```
WSABUF buf[3];  
Buf[0].buf = &move_p;   buf[0].len = sizeof(move_p);  
Buf[1].buf = &attack_p; buf[1].len = sizeof(attack_p);  
Buf[2].buf = &chat_p;   buf[2].len = sizeof(chat_p);  
WSASend(s, buf, 3);
```

Windows Network

- Socket 끝기

- `int closesocket(SOCKET s)`

- `s` : 소켓

- return value : 0 on success

Windows Network

- PORT 묶기

- 포트를 선점해서 다른 프로그램이 사용하지 못하도록 한다.

- `int bind(SOCKET s, const struct sockaddr* name, int namelen)`

- s : 소켓

- name, namelen : 상대 소켓 주소, 주소 길이

- INADDR_ANY로 전체 주소로부터의 접속을 허용

- `info.sin_addr.S_in.S_addr = htonl(INADDR_ANY)`

Windows Network

- Socket 접속 대기
 - 소켓이 접속을 받을 수 있도록 만든다.
 - `int listen(SOCKET s, int backlog)`
 - s : 소켓
 - backlog : 접속 대기 큐의 최대 연결 가능 숫자
 - 서버에 도착한 후 `accept`될 때 까지 기다리고 있는 소켓연결의 최대 개수
 - 싱글 스레드 프로그램에서는 5정도가 적당
 - 대용량 서버는 20-200, 또는 **SOMAXCONN**

Windows Network

- 연결 Socket 생성

- 연결된 소켓을 만든다.
- SOCKET WSAAccept(SOCKET s, struct sockaddr* name, LPINT addrlen, LPCONDITIONPROC lpfnCondition, DWORD dwCallbackData)
 - s : listen을 하고 있는 소켓
 - name, addrlen : 연결된 상대 소켓의 주소
 - lpfnCondition : 연결 거절을 판단하는 함수
 - 보통 NULL
 - dwCallbackData : lpfnCondition에 들어갈 값
 - Return Value : 데이터 전송용 소켓

간단한 실습

- Echo Sever

- 클라이언트가 입력한 문장을 그대로 반사

간단한 실습 : 클라이언트

```
#include <iostream>
#include <WS2tcpip.h>
using namespace std;
#pragma comment (lib, "WS2_32.LIB")

const char* SERVER_ADDR = "127.0.0.1";
const short SERVER_PORT = 4000;
const int BUFSIZE = 256;

int main()
{
    wcout.imbue(locale("korean"));
    WSADATA WSAData;
    WSASStartup(MAKEWORD(2, 0), &WSAData);
    SOCKET s_socket = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, 0, 0, 0);
    SOCKADDR_IN server_addr;
    ZeroMemory(&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    inet_pton(AF_INET, SERVER_ADDR, &server_addr.sin_addr);
    connect(s_socket, reinterpret_cast<sockaddr*>(&server_addr), sizeof(server_addr));
    for (;;) {
        char buf[BUFSIZE];
        cout << "Enter Message : "; cin.getline(buf, BUFSIZE);
        DWORD sent_byte;
        WSABUF mybuf;
        mybuf.buf = buf;          mybuf.len = static_cast<ULONG>(strlen(buf)) + 1;
        WSASend(234, &mybuf, 1, &sent_byte, 0, 0, 0);

        char recv_buf[BUFSIZE];
        WSABUF mybuf_r;
        mybuf_r.buf = recv_buf;    mybuf_r.len = BUFSIZE;
        DWORD recv_byte;
        DWORD recv_flag = 0;
        WSARcv(s_socket, &mybuf_r, 1, &recv_byte, &recv_flag, 0, 0);
        cout << "Server Sent [" << recv_byte << "bytes] : " << recv_buf << endl;
    }
    WSACleanup();
}
```

간단한 실습 : 서버

```
#include <iostream>
#include <WS2tcpip.h>
using namespace std;
#pragma comment (lib, "WS2_32.LIB")

const short SERVER_PORT = 4000;
const int BUFSIZE = 256;

int main()
{
    WSADATA WSAData;
    WSASStartup(MAKEWORD(2, 0), &WSAData);
    SOCKET s_socket = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, 0, 0, 0);
    SOCKADDR_IN server_addr;
    ZeroMemory(&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(s_socket, reinterpret_cast<sockaddr*>(&server_addr), sizeof(server_addr));
    listen(s_socket, SOMAXCONN);

    INT addr_size = sizeof(server_addr);
    SOCKET c_socket = WSAAccept(s_socket, reinterpret_cast<sockaddr*>(&server_addr), &addr_size, 0, 0);
    for (;;) {
        char recv_buf[BUFSIZE];
        WSABUF mybuf;
        mybuf.buf = recv_buf;    mybuf.len = BUFSIZE;
        DWORD recv_byte;
        DWORD recv_flag = 0;
        WSARcv(c_socket, &mybuf, 1, &recv_byte, &recv_flag, 0, 0);
        cout << "Client Sent [" << recv_byte << "bytes] : " << recv_buf << endl;

        DWORD sent_byte;
        mybuf.len = recv_byte;
        WSASend(c_socket, &mybuf, 1, &sent_byte, 0, 0, 0);
    }
    WSACleanup();
}
```

디버깅 팁

● 네트워크 관련 에러 검출

```
void error_display(const char *msg, int err_no )
{
    WCHAR *lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM,
        NULL, err_no,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
        (LPTSTR)&lpMsgBuf, 0, NULL );
    std::cout << msg;
    std::wcout << L"에러 " << lpMsgBuf << std::endl;
    while(true);
    LocalFree(lpMsgBuf);
}
```

디버깅 팁

- 한글이 나오지 않는데???

- 다음 추가

```
std::wcout.imbue(std::locale("korean"));
```

- (2021년 이전 설치된 Visual Studio의 경우)

- VisualStudio -> 솔루션탐색기 -> 프로젝트 -> 오른쪽클릭 -> 속성 -> 구성속성 -> 고급 -> 프로젝트 기본값 -> 문자 집합 -> 유니코드 문자 집합 사용

IPv6

- 현재 사용하고 있는 IPv4는 네트워크 주소가 모자람.
 - 32bit = 40억
- IPv6가 새로운 표준으로
 - 128비트 : 우주에 존재하는 모든 원자의 개수 보다 큰 숫자. 통 크게 해결.
- 하지만 아직 옛날 기계들과 옛날 SW들은 IPv4만 인식
- 적극 보급 정책 실행 중
 - Apple의 앱스토어와 구글의 PlayStore는 IPv6를 구현해야 등록 가능
 - 우리나라가 뒤떨어져 있음.
- IPv4와 다른 프로토콜을 사용하기 때문에 서로 통신 안됨
 - 소켓 프로그래밍 API는 같음, Protocol만 다름.

IPv6

- 프로그래밍

- AF_INET => AF_INET6
- WSA Verseion : 2.0 => 2.2
- SOCKADDR_IN => SOCKADDR_IN6
- INADDR_ANY => in6addr_any
- “127.0.0.1” => “::1” 또는 “0:0:0:0:0:0:0:1”

- 문제

- 옛날 공유기를 사용한다면 다른 컴퓨터와 연결이 안될 수 있다...
- 이후 실습에서는 IPv4를 사용.

숙제 (#2)

- 게임 서버/클라이언트 연동 구현
 - 내용
 - 숙제 (#1)의 프로그램을 Client/Server 모델로 분리
 - Client :
 - 키입력을 받아서 서버 프로그램에 보낸다
 - 서버에서 보내온 좌표로 말을 이동시킨다.
 - 시작할 때 서버의 IP주소를 입력 받는다.
 - Server
 - 클라이언트에서 보내온 키입력을 보고 말의 위치 변경
 - 변경된 위치를 클라이언트에 전송
 - 제약
 - Windows에서 Visual Studio 2022/2019로 작성 할 것
 - 그래픽의 우수성을 보는 것이 아님 (unity와 언리얼 사용금지)
 - 제출
 - Zip으로 소스를 묶어서 제출
 - 컴파일 및 실행 가능할 것, 중간 파일 제거(.exe파일 x64, Release, .vs폴더 같은 것 제외!)
 - E-Class의 숙제 시스템을 통해 제출