1. 네트워크와 소켓 프로그래밍
   * 응용 프로그램이 통신하려면 결정되어야 하는 요소
     + 사용할 프로토콜
     + 송.수신 측 IP 주소와 포트 번호
   * 데이터 타입으로 볼 수 있고 통신의 종단점 인 것은? 소켓
2. 윈도우 소켓 시작하기
   * WSAGetLastError(): 구체적인 오류코드 얻을 수 있음
   * FormatMessage(): 오류코드에 대응하는 오류 메시지
   * WSAStartup(): 프로그램에서 사용할 윈속 버전을 요청해 윈속 라이브러리를 초기화
   * WSACleanup(): 윈속 사용을 중지함을 운영체제에 알리고, 관련 리소스를 반환한다.
   * socket(): 소켓 생성
   * closesocket(): 소켓 닫기. 해당 리소스 반환
3. 소켓 주소 구조체 다루기
   * SOCKADDR: 기본타입으로, 다양한 소켓 함수의 매개변수 타입으로 사용. 소켓 주소 구조체.
   * IN\_ADDR: IP 주소를 저장하기 위한 구조체
   * 빅 엔디안: 최상위 바이트부터 차례로 저장하는 방식
   * 리틀 엔디안: 최하위 바이트부터 차례로 저장하는 방식
   * htons(), htonl(): 호스트 바이트 정렬 -> 네트워크 바이트 정렬
   * ntohs(), ntohl(): 네트워크 바이트 정렬 -> 호스트 바이트 정렬
   * inet\_addr(): 문자열 -> 숫자
   * inet\_ntoa(): 숫자 -> 문자열
   * WSAStringToAddress(): 문자열 -> 숫자
   * WSAAddressToString(): 숫자 -> 문자열
   * Gethostbyname(): 도메인 이름 -> IPv4/IPv6 주소(네트워크 바이트 정렬)
   * Gethostbyaddr(): IPv4/IPv6 주소(네트워크 바이트 정렬) -> 도메인 이름
4. TCP 서버-클라이언트
   * TCP 서버 함수
     + socket(): 사용할 프로토콜 결정
     + bind(): 지역 IP, 포트번호 결정
     + listen(): TCP를 LISTENING 상태로 바꾼다.
     + accept(): 원격 IP, 포트번호 결정
     + recv(), send(): 데이터 송수신 함수
     + closesocket(): 5으로 통신 수행 후 소켓 닫음
   * TCP 클라이언트 함수
     + socket(): 사용할 프로토콜 결정
     + connect(): 원격, 지역 IP, 포트번호 결정. bind() 함수의 역할도 함.
     + send(), recv(): 데이터 송수신 함수
     + closesocket(): 3으로 통신 수행 후 소켓 닫음

**※ send() 함수 사용 시 주의할 점**

1. send() 함수가 리턴했다고 실제 데이터가 전송된 것은 아니며, 일정 시간이 지나야만 하부 프로토콜을 통해 전송이 완료된다.
2. TCP의 경우, 수신버퍼에 들어온 데이터가 송신버퍼에 발송한 데이터의 전부가 아닐 수 있다. TCP 프로토콜은 응용 프로그램이 보낸 데이터의 경계를 구분하지 않기 때문이다.
3. 검증작업 후 제대로 데이터가 전달되지 않았다면 운영체제의 송신버퍼에 재요청. 즉, 데이터를 확인 후 삭제해야 함.
   * 블로킹 소켓: 블로킹 소켓을 대상으로 send() 함수를 호출하면, 송신 버퍼의 여유 공간이 send() 함수의 세 번째 인자인 len보다 작을 경우 해당 프로세스는 대기 상태가 된다. 송신버퍼에 충분한 공간이 생기면 프로세스는 깨어나고 len 크기만큼 데이터 복사가 일어난 후 send() 함수가 리턴한다. 이 경우 send() 함수의 리턴값은 len이다.
   * recv()의 성공적인 리턴
     + 수신 버퍼에 데이터가 도달한 경우
     + 접속이 정상 종료한 경우

**※ recvn() 함수 사용하는 이유**

* recv() 세 번째 인자인 len으로 지정한 크기보다 적은 데이터가 버퍼에 복사될 수 있다는 점에 기인한다. 이는 TCP가 데이터 경계를 구분하지 않기 때문이다. 따라서 자신이 받을 데이터의 크기를 미리 안다면 그 만큼 recv() 함수를 여러 번 호출해야 하기 때문에 recvn() 함수를 통해 이를 처리한다.
  + 소켓 버퍼
    - 송신 버퍼: 데이터를 전송하기 전에 임시로 저장해두는 영역
    - 수신 버퍼: 받은 데이터를 응용 프로그램이 처리하기 전까지 임시로 저장해두는 영역

1. 데이터 전송하기
   * 경계 구분을 위한 네 가지 방식
     + 고정: 항상 고정길이만 주고받음
     + 가변+EOR: EOR 나올때까지 데이터 읽음
     + 고정+가변: 데이터의 크기 고정으로, 읽은 크기만큼 가변 길이 데이터 읽음
     + 가변 길이 데이터 전송 후 접속 종료: recv() 리턴 값이 0이 될 때까지 데이터 읽음
   * 데이터 전송 시 고려 사항
     + 바이트 정렬
     + 구조체 멤버 맞춤: 메모리의 시작 주소를 결정하는 컴파일러 규칙. 메모리 낭비를 방지. #pragma pack()를 사용함. #pragma pack(1)은 구조체 멤버 맞춤을 1바이트 경계로 변경
2. 멀티스레드
   * 교착상태: 서버와 클라이언트의 send(), recv() 함수 호출 순서가 서로 맞지 않아 영원히 일어나지 않을 사건을 두 프로세스가 기다리는 상황
   * 윈도우의 프로세스
     + 프로세스: 정적
     + 스레드: 동적
   * 컨텍스트 전환: 스레드의 실행 상태의 저장과 복원 작업
   * CreateThread(): 스레드를 생성하는 함수
   * 윈도우에서 스레드를 종료하는 방법
     + 스레드 함수가 리턴한다.
     + 스레드 함수 안에서 ExitThread() 함수 호출
     + 다른 스레드가 TerminateThread() 함수를 호출해 강제 종료
     + 주 스레드 종료
   * 스레드 스케줄링(CPU 스케줄링): CPU 시간을 적절히 분배하기 위한 정책
     + 프로세스 우선순위: 우선순위 클래스 (한 프로세스가 생성한 스레드는 우선순위가 모두 같음)
     + 스레드 우선수위: 우선순위 레벨(같은 프로세스에 속한 스레드 간 상대적인 우선순위 결정)
     + 기본 우선순위: 프로세스, 스레드 우선순위를 결합 -> 스레드 스케줄링에 사용됨
     + SetThreadPriority(), GetThreadPriority()
   * WaitForSingleObject(): 특정 스레드가 종료할 때까지 기다리기
   * WaitForMultipleObjects(): 여러 스레드가 종료할 때까지 기다리기
   * SuspendThread(): 스레드 일시 중지
   * ResumeThread(): 스레드 재시작
   * SuspendThread, ResumeThread는 중지횟수 관리해야함. 정지 2번 호출-> 재개 2번 호출
   * getpeername(): 소켓을 통해 원격 IP 주소, 포트 번호 얻음
   * getsockname(): 소켓을 통해 지역 IP 주소, 포트 번호 얻음
   * 스레드 동기화
3. 임계 영역: 동기화 객체는 아님
   * 유저 메모리 영역에 존재하는 단순한 구조체
   * 공유 자원을 보호하기는 하지만, 접근 순서 제어는 못 함.
   * InitializeCriticalSection(), DeleteCriticalSection()
   * EnterCriticalSection(), LeaveCriticalSection(): 공유자원 접근하기 전후 사용
4. 이벤트: 사건 발생을 다른 스레드에 알리는 동기화 기법. 동기화 객체임
   * 접근 순서 제어 가능
   * CreateEvent(): 이벤트 생성 함수
   * SetEvent(): 비신호->신호, ResetEvent(): 신호->비신호
   * 자동 리셋 이벤트: 기다리는 스레드 중 하나만 깨움. ResetEvent() 불필요
   * 수동 리셋 이벤트: 기다리는 스레드 전부 깨움. ResetEvent() 호출 필요
5. UDP 서버-클라이언트
   * TCP와 UDP의 공통점
     + 포트번호를 이용해 주소 지정
     + 데이터 오류 체크(체크섬 이용)

**※ TCP와 달리 UDP가 신뢰성 있는 데이터를 전달하지 못하는 이유**

* UDP에서 데이터 재전송과 데이터 순서 유지 작업을 하지 않기 때문이다. UDP는 도착한 데이터에 오류가 있다고 판단하면 이 데이터를 그대로 삭제해버린다. 따라서 송신버퍼도 없다.
  + 텍스트이(가) 표시된 사진

    자동 생성된 설명UDP 서버
    - socket(): 소켓 생성, 프로토콜 결정
    - bind(): 지역 IP, 포트번호 결정
    - recvfrom(): 원격IP, 포트번호 알 수 있음
    - sendto(): 데이터 보내기
    - closesocket()
  + UDP 클라이언트
    - socket(): 소켓 생성, 프로토콜 결정
    - sendto(): 원격, 지역 IP, 포트번호 알 수 있음. UDP에서 사용할 경우 커널영역에 복사됨.
    - recvfrom(): 데이터 받기. TCP와 달리 데이터 경계 구분 작업이 필요 없다.
    - closesocket()
  + 클라에서 send, recv 함수를 사용하는 UDP 서버-클라 모델
    - sendto() 함수 사용할 때보다 효율적. connect() 함수로 서버 주소를 한 번만 설정해두면 send() 함수가 재사용
    - 데이터 받은 후 송신자의 주소 확인 안해도됨. recv()는 connect()로 설정한 대상이 보낸 데이터만 수용하기 때문.
  + 브로드 캐스팅: 1대 다 통신

과제 정리

* + - 1. 과제 1 – 윈속 1.1 버전을 사용하도록 예제 수정

1. LOBYTE(): 하위 8비트만 알 수 있음. 0과 & 연산을 한다.
2. HIBYTE(): 상위 8비트만 알 수 있음. 쉬프트 연산을 통해 8비트 이동시킨다.
   * + 1. 과제 2 – 내 컴퓨터의 바이트 정렬 방식 알아내기

* htons()을 사용해서 넣은 값과 리턴 값이 다르면 시스템은 리틀엔디안. 같으면 빅엔디안
* ntohs()를 사용하면 안 되는 이유

: ntohs()에 들어오는 인자는 빅엔디안이라고 생각하므로.

* + - 1. 과제 3 – “LISTENING” 상태인 포트번호 모두 알아내기
    - for문 돌려서 정해진 범위 내 포트번호로 소켓 생성, 통신 반복하여 통신이 되면 LISTENING.
    - netstat -a-n: 포트번호의 상태를 확인하는 명령어
    1. 과제 4 – 동영상 파일 보내기 (고정 + 가변)
    - recv() 와 recvn()의 차이  
      : recv() 함수 사용시 len보다 적은 데이터가 버퍼에 복사될 수 있다. (TCP가 데이터 경계를 구분하지 않아서) 따라서 데이터의 크기를 미리 알고 있다면 recv() 함수를 그 만큼 받을 때까지 여러 번 호출해야 함.

5. 과제 5 – 멀티스레드를 이용해 파일 보내기

* + - 멀티스레드 2개 생성하여 동시에 두 개의 클라이언트가 파일을 보내도록 함.