Qt_4일차_보고서_20기_인턴_2025402040_컴퓨터정보공학부_노기문

-hw_02 : udp 통신보고서 (카카오톡 구현을 통해) -

<목차>

<UDP 통신>

통신(Communication)은 현대 사회와 기술 시스템의 핵심적인 기능으로, 두 개 이상의 개체가 정보를 주고받으며 상호작용하는 과정을 의미한다. 이는 단순히 데이터를 전송하는 행위에 그치지 않고, 전달된 정보를 해석하여 의미를 공유하는 단계까지 포함한다. 따라서 효과적인 통신은 정보의 전달과 의미 해석이 동시에 이루어져야만 성립된다.

통신은 "두 개 이상의 개체가 데이터를 주고받는 과정"으로 정의된다. 여기서 개체란 사람, 기계, 혹은 네트워크 장비 등 물리적 혹은 논리적 시스템을 모두 포함한다. 이 과정은 단순한 데이터 이동이 아니라, 정보 전달 + 의미 해석이 결합되어야 한다. 즉, 송신된 데이터가 수신자에게 단순히 도달하는 것을 넘어, 수신자가 그 의미를 이해하고 활용할 수 있어야 진정한 통신이 이루어진다고 할 수 있다.
통신 시스템은 일반적으로 다음 네 가지 기본 요소로 구성된다.

송신기(Transmitter)
데이터를 생성하고 신호로 변환하여 전송하는 장치이다. 예를 들어, 사람의 음성을 전기 신호로 바꾸는 마이크, 혹은 컴퓨터에서 데이터를 패킷으로 변환하는 네트워크 카드가 이에 해당한다.

수신기(Receiver)
송신기로부터 전달된 신호를 받아 해석하고 원래의 데이터로 복원하는 장치이다. 예를 들어, 스피커는 전기 신호를 다시 음성으로 변환하며, 네트워크 장비는 패킷을 해석하여 응용 프로그램에 전달한다.

전송 매체(Channel)
송신기와 수신기 사이에서 데이터를 전달하는 물리적·논리적 경로이

다. 이는 유선(전선, 광섬유)일 수도 있고 무선(라디오파, 적외선, 위성 통신)일 수도 있다. 전송 매체는 통신의 신뢰성과 속도에 직접적인 영향을 미친다.

프로토콜(Protocol, 규칙)

송신자와 수신자가 올바르게 데이터를 교환하기 위해 지켜야 하는 규칙과 약속이다. 예를 들어, TCP/IP는 인터넷 통신의 대표적인 프로토콜이며, 데이터의 전송 방식, 오류 검출, 재전송 절차 등을 정의한다.

통신은 단순히 데이터를 주고받는 것이 아니라, 의미를 공유하고 해석하는 복합적인 과정이다. 송신기, 수신기, 전송 매체, 프로토콜이라는 네 가지 요소가 유기적으로 작동할 때 원활한 정보 교환이 가능하다. 따라서 통신의 본질은 기술적 신호의 교환을 넘어, 의미 있는 상호작용과 이해를 실현하는 데 있다.

통신방식

통신은 단순히 데이터를 주고받는 과정이 아니라, 그 방식에 따라 성능, 효율성, 신뢰성이 크게 달라진다. 본 보고서에서는 통신 방식을 전송 방향과 회선 구성 방식으로 나누어 살펴본다. 이를 통해 실제 통신 시스템의 구조와 특징을 이해할 수 있다.

2. 전송 방향에 따른 분류

단방향(Simplex)

정의: 한쪽 방향으로만 데이터 전송이 가능한 방식.

특징: 송신자는 오직 데이터를 보내고, 수신자는 오직 데이터를 받는다.

예시: TV 방송, 라디오 방송.

장점: 구조가 단순하고 비용이 적게 든다.

단점: 상호작용 불가, 피드백 전달 불가능.

반이중(Half Duplex)
정의: 양방향 통신이 가능하나, 동시에 송수신은 불가능한 방식.
특징: 송신과 수신은 교대로 이루어진다.
예시: 무전기 통신(“오버”라고 말해야 교신 가능).

장점: 단방향보다 효율적, 상호작용 가능.
단점: 동시에 송수신 불가로 인해 지연 발생 가능.

전이중(Full Duplex)
정의: 양방향 통신이 동시에 이루어지는 방식.
특징: 송신과 수신이 동시에 가능하다.
예시: 전화 통화.

장점: 자연스러운 대화 가능, 효율성 높음.
단점: 구현이 복잡하고 비용이 증가할 수 있음.

3. 회선 구성 방식에 따른 분류
회선 교환(Circuit Switching)
정의: 통신 경로를 먼저 전용으로 연결한 후, 그 회선을 통해 통신하는 방식.
특징: 연결이 성립되면 통화 중에는 회선이 고정적으로 점유된다.
예시: 유선 전화망.

장점: 연결 중에는 안정적이고 지연이 적다.
단점: 회선을 독점하므로 자원 낭비 발생 가능.

패킷 교환(Packet Switching)
정의: 데이터를 작은 단위인 패킷으로 나누어 전송하는 방식.
특징: 각 패킷은 독립적으로 전달되며, 목적지에서 재조립된다.
예시: 인터넷, IP 기반 통신.

장점: 회선 자원을 효율적으로 공유 가능, 다수 사용자 동시 지원.
단점: 패킷 손실, 지연, 순서 변경 등이 발생할 수 있음.

통신 방식은 데이터의 전송 방향과 회선 구성 방식에 따라 크게 달라진다. 단방향, 반이중, 전이중의 구분은 통신의 상호작용 수준을 결정하며, 회선 교환과 패킷 교환의 구분은 자원 활용 방식과 효율성을 좌우한다. 현대 사회에서는 인터넷과 같은 패킷 교환 + 전이중 방식이 주로 활용되며, 이는 빠르고 효율적인 통신 환경을 제공한다.

통신프로토콜

효과적인 통신은 단순히 신호의 교환만으로는 이루어지지 않는다. 송신자와 수신자가 동일한 규칙과 약속을 공유해야 데이터가 정확하고 효율적으로 전달될 수 있다. 이러한 규칙과 약속을 총칭하여 **통신 프로토콜(Communication Protocol)**이라 한다.
통신 프로토콜이란 통신을 원활히 하기 위해 정해진 규칙과 약속을 의미한다. 이는 데이터의 형식, 전송 순서, 오류 처리 방식 등을 포함하며, 네트워크 장치 간 상호 호환성을 보장한다.
대표적인 예로는 TCP/IP, HTTP, FTP, Bluetooth 프로토콜 등이 있으며, 이들은 각각 특정 환경에서 통신의 표준을 제시한다.

주요 기능
주소 지정(Addressing)

기능: 누구와 통신할지를 구분.

설명: 네트워크 상에서 송수신 대상을 정확히 식별해야 한다. 예를 들어, IP 주소나 MAC 주소는 주소 지정의 대표적인 수단이다.

에러 제어(Error Control)

기능: 전송 과정에서 발생할 수 있는 오류를 검출하고 복구.

설명: 전송 도중 데이터 손상, 패킷 손실 등이 발생할 수 있다. 이를 CRC, 체크섬, 자동 재전송 요청(ARQ) 같은 기술을 통해 해결한다.

흐름 제어(Flow Control)

기능: 송신기와 수신기 간 속도 조절.

설명: 송신자가 너무 빠르게 데이터를 보내면 수신자가 처리하지 못하고 손실이 발생한다. 따라서 슬라이딩 윈도우, 스톱 앤 웨이트 같은 기법을 사용해 균형을 맞춘다.

동기화(Synchronization)

기능: 데이터 순서와 타이밍 맞추기.

설명: 수신자가 데이터의 순서를 잘못 해석하거나 시점을 놓치면 통신이 왜곡된다. 이를 위해 클럭 동기화, 시퀀스 번호, 시간 동기화 프로토콜(NTP) 등이 활용된다.
통신 프로토콜은 단순한 규칙이 아니라 통신의 안정성과 효율성을 보장하는 핵심 요소이다. 주소 지정, 에러 제어, 흐름 제어, 동기화와 같은 기능은 네트워크 환경에서 데이터가 올바르게 전달되고 해석될 수 있도록 지원한다. 오늘날 인터넷을 비롯한 모든 통신 체계는 이러한 프로토콜 위에서 안정적으로 운영되고 있다.

통신 기술은 물리적 매체의 사용 여부에 따라 유선(Wired) 통신과 무선(Wireless) 통신으로 구분된다. 두 방식은 각각의 특성과 장단점에 따라 다양한 환경에서 선택적으로 활용된다. 본 보고서에서는 유선과 무선 통신의 특징을 비교하고, 실제 사용 시 고려해야 할 선택 기준을 제시한다.

유선 통신(Wired Communication)
정의: 물리적 케이블을 통해 데이터를 전송하는 방식.

주요 형태: LAN 케이블(Ethernet), 광케이블(Fiber Optics) 등.
특징:

안정성이 높아 외부 간섭에 강함.
고속 데이터 전송 가능.
설치 시 물리적 제약이 크며 이동성이 떨어짐.
활용 예시: 사무실 네트워크, 데이터 센터, 방송 장비 연결.

무선 통신(Wireless Communication)
정의: 전파를 통해 데이터를 송수신하는 방식.
주요 형태: Wi-Fi, Bluetooth, 5G, 위성 통신 등.

특징:

이동성과 편의성이 뛰어나 장소 제약이 적음.
설치 비용이 낮고 다양한 기기와 쉽게 연결 가능.
전파 간섭, 보안 문제, 전송 속도의 변동 가능성이 존재.

선택 기준
유선과 무선 중 어떤 방식을 선택할지는 다음과 같은 요소에 의해 결정된다.

속도: 대용량 데이터를 빠르게 전송해야 한다면 유선이 적합.
안정성: 끊김 없는 연결과 보안이 중요한 환경에서는 유선 선호.
범위: 넓은 지역을 커버하거나 이동성을 고려한다면 무선 선택.
비용: 설치와 유지 비용을 종합적으로 비교해야 함.

유선 통신은 안정성과 속도를 강점으로 하며, 무선 통신은 편의성과 이동성에서 우위를 가진다. 따라서 실제 환경에서는 두 방식을 적절히 혼합하여 사용하는 경우가 많다. 예를 들어, 기업 내부망은 유선을 사용해 보안을 강화하고, 사용자 단말기 연결은 무선으로 제공하는 방식이 대표적이다.
UDP

네트워크 통신에서 전송 계층(OSI 7계층 중 4계층)은 응용 프로그램 간 데이터를 전달하는 핵심 역할을 한다. 이 계층에서 사용되는 대표적인 프로토콜은 TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)이다. 본 보고서에서는 UDP의 정의, 특징, 그리고 활용 목적을 살펴본다.

UDP는 OSI 7계층 모델의 전송 계층(4계층)에 속하는 통신 프로토콜이다. TCP와 달리 연결을 확립하지 않고 데이터를 전송하며, 가볍고

빠른 전송을 목표로 설계되었다. 따라서 "비연결형(Connectionless)"과 "비신뢰성(Unreliable)"의 특성을 가지며, 대신 낮은 지연과 빠른 전송 속도를 제공한다.

UDP의 주요 특징

규약 비연결성(Connectionless)

송신자는 수신자와 연결 절차 없이 데이터를 전송한다.
연결 설정에 드는 시간이 없으므로 즉각적인 통신 가능.

비신뢰성(Unreliable)

패킷이 손실되거나 순서가 바뀌더라도 자동 복구 기능이 없다.
신뢰성을 보장해야 하는 경우, 응용 계층에서 별도의 보완 장치 필요.

빠른 전송(Fast Transmission)

헤더가 간단하고 오버헤드가 적어 TCP보다 속도가 빠르다.
실시간 성능이 중요한 환경에서 사용됨.

활용 사례

UDP는 신뢰성보다 속도가 중요한 응용 분야에서 주로 사용된다.

스트리밍 서비스: 영상/음성 실시간 전송(예: YouTube, IPTV).
온라인 게임: 빠른 반응 속도를 위해 패킷 일부 손실을 허용.
VoIP(Voice over IP): 인터넷 전화에서 지연 최소화.
DNS(Domain Name System): 간단한 요청-응답 구조에 적합.

UDP는 전송 계층에서 빠른 속도를 보장하기 위해 설계된 경량 프로토콜이다. 비연결성과 비신뢰성이라는 제약이 있지만, 이는 실시간성과 효율성이 중요한 응용에서 장점으로 작용한다. 따라서 TCP가 신

뢰성 중심의 프로토콜이라면, UDP는 속도와 실시간성을 위한 프로토콜이라 할 수 있다.

OSI 7계층 모델과 UDP의 위치 및 역할

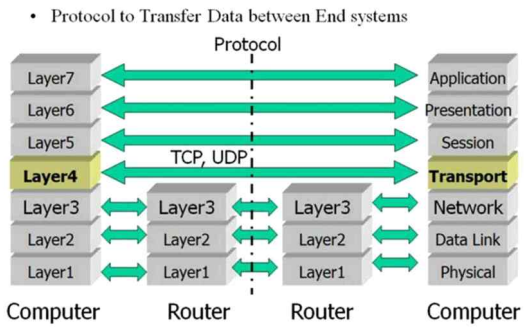
네트워크 통신을 설명하는 대표적인 이론적 구조가 OSI 7계층(Open Systems Interconnection Model)이다. 이 모델은 통신 과정을 7개의 계층으로 나누어 각 계층이 담당하는 역할을 명확히 구분한다. UDP(User Datagram Protocol)는 이 중 **전송 계층(4계층)**에 속하는 핵심 프로토콜로, 빠른 데이터 전송을 위해 설계되었다.

OSI 7계층 모델 개요

UDP

ROBIT ROBOT SPORT GAME TEAM

- OSI 7계층 중 전송 계층(4계층)에 속하는 통신
- 규약비연결성 & 비신뢰성 특징을 가지며, 빠른 전송을 위해 설계됨



물리 계층(Physical Layer)

신호의 실제 전송 담당 (케이블, 무선 전파, 하드웨어).

데이터 링크 계층(Data Link Layer)

MAC 주소 기반으로 프레임을 전송, 오류 검출 수행.

네트워크 계층(Network Layer)

IP 주소 기반으로 경로를 설정하고 라우팅 수행.

전송 계층(Transport Layer)

데이터의 종단 간 전달 담당. TCP와 UDP가 대표적인 프로토콜.

세션 계층(Session Layer)

통신 세션의 생성, 유지, 종료 관리.

표현 계층(Presentation Layer)

데이터 형식 변환, 압축, 암호화 처리.

응용 계층(Application Layer)

사용자가 직접 접하는 서비스 (HTTP, FTP, 이메일, 영상 앱 등).

UDP의 역할 (전송 계층 프로토콜)

UDP는 전송 계층에서 다음과 같은 기능을 수행한다.

하위 계층(IP 계층)으로부터 지원을 받아, 목적지까지 데이터를 전달.

상위 계층(응용 계층)으로부터 받은 데이터를 빠르게 송출.

연결 절차 없이 전송이 가능하며, 빠른 속도를 제공.

헤더 구조가 단순하여 지연 시간이 적음.

즉, UDP는 속도와 단순성을 최우선으로 하는 전송 프로토콜이다.
OSI 7계층 모델은 네트워크 구조를 이해하는 기본 틀을 제공하며, UDP는 전송 계층에서 빠른 데이터 전달을 담당한다. TCP가 신뢰성과 안정성을 보장하는 반면, UDP는 효율성과 속도를 강조한다. 따라서 영상 스트리밍, 온라인 게임, 실시간 서비스와 같이 지연 최소화

가 중요한 분야에서 널리 사용된다.

UDP의 세부 특징과 활용 분야

UDP(User Datagram Protocol)는 TCP와 함께 전송 계층에서 사용되는 대표적인 프로토콜이다. UDP는 속도와 단순성을 우선시하며, 데이터 전송 단위인 데이터그램(Datagram)을 사용한다. 본 보고서에서는 UDP의 세부 구조와 특성을 정리하고, 이를 실제 응용 사례와 연결하여 설명한다.

데이터 전송 단위: 데이터그램(Datagram)

각 데이터그램은 독립적으로 처리되며, 서로 다른 경로를 통해 전송될 수 있다.
연결 과정 없이 전송이 가능해 빠른 통신 제공.

연결 과정 없음(Connectionless)

수신자가 준비되어 있지 않아도 송신자는 데이터를 보낼 수 있다.
연결 설정과 해제 절차가 없어 지연 시간이 최소화됨.

신뢰성 부족(Unreliable)

오류 검출은 Checksum으로 최소한만 제공.
데이터 손실, 순서 뒤바뀜, 중복 전송에 대한 보장 없음.

고정 헤더 구조(8바이트)

출발지 포트 번호
목적지 포트 번호
길이(Length)

체크섬(Checksum)

→ 단순하고 효율적인 구조로, TCP보다 훨씬 작은 오버헤드를 가짐.

속도와 효율성

연결 절차와 복잡한 오류 제어가 없으므로 전송 속도가 매우 빠름.

대신 신뢰성 있는 데이터 전송을 요구하는 경우에는 부적합.

UDP의 한계

데이터 순서 보장 없음.

패킷 손실 시 재전송 기능 없음.

네트워크 품질에 따라 통신 성능이 불안정할 수 있음.

활용 분야

UDP는 신뢰성보다 속도가 중요한 상황에서 주로 활용된다.

영상 스트리밍 서비스: 약간의 데이터 손실을 감수하고 실시간성을 확보.

온라인 게임: 빠른 반응 속도를 위해 손실된 패킷을 무시하고 최신 정보 유지.

VoIP(인터넷 전화): 음성 지연 최소화.

센서 데이터 전송: IoT 기기에서 단순한 데이터 전달에 적합.

UDP는 단순성, 속도, 실시간성을 최우선으로 하는 통신 프로토콜이다. 신뢰성 부족이라는 한계가 있지만, 스트리밍, 게임, 실시간 제어 등에서는 오히려 장점으로 작용한다. 따라서 현대 네트워크 환경에서 TCP와 상호 보완적으로 사용되며, 특정 요구사항에 따라 선택적으로 적용된다.

로봇 영상 데이터 전송

특징: 로봇 영상 데이터는 초당 15~30fps 수준으로 대용량 이미지를 연속적으로 전송해야 한다.

요구사항: 지연이 짧고 빠른 전송 속도가 필수적이다.

UDP의 적합성: 연결 설정 절차가 없고 헤더가 단순하여 지연 최소화에 유리하다. 따라서 로봇 영상 데이터 전송에 효과적이다.

데이터 손실 허용성

영상이나 음성 스트리밍의 경우 일부 데이터 손실이 발생해도 다음 프레임으로 보완이 가능하다.

즉, 완벽한 신뢰성보다 전송 속도가 더 중요한 서비스에서는 UDP가 적합하다.

예: CCTV 실시간 모니터링, 로봇 원격 제어 영상 스트리밍.

4. 중요한 데이터 전송 시 보완 방법

상태 메시지, 명령어 등 손실되면 치명적인 데이터는 단순 UDP로는 부족하다.

이 경우 다음과 같은 보완 방법이 사용된다:

UDP + CRC8: 간단한 오류 검출 코드를 추가하여 신뢰성 확보.

TCP와 병행 사용: 제어 신호나 명령은 TCP로, 영상 데이터는 UDP로 전송하는 혼합 구조.

UDP는 대용량, 실시간, 빠른 전송이 요구되는 응용 분야에 최적화된 프로토콜이다. 특히 로봇 영상 데이터와 같은 환경에서는 일부 손실

을 감수하더라도 속도가 중요하므로 UDP가 적합하다. 그러나 제어 메시지와 같은 중요한 데이터는 보완 기법(CRC8, TCP 병행)을 통해 신뢰성을 강화해야 한다. 따라서 UDP는 속도와 신뢰성 요구사항을 구분하여 선택적으로 적용해야 하는 프로토콜이라 할 수 있다.

Qt 기반 UDP 소켓 프로그래밍 구현 과정

UDP(User Datagram Protocol)는 빠르고 단순한 데이터 전송 방식을 제공한다. Qt 프레임워크에서는 QUdpSocket 클래스를 통해 손쉽게 UDP 통신을 구현할 수 있으며, 로봇 제어, 실시간 영상 전송, 센서 데이터 교환과 같은 응용에서 널리 활용된다. 본 보고서에서는 QUdpSocket을 활용한 UDP 통신 구현 과정을 단계별로 설명한다.

UDP 포트 번호 체계
UDP 포트 번호는 0~65535 범위에서 사용된다.

0~1023번 (Well-known Port): 시스템 주요 서비스 예약 (예: HTTP 80, FTP 21).

1024~49151번 (Registered Port): 일반 프로그램 및 사용자 애플리케이션 사용.

49152~65535번 (Dynamic/Private Port): 클라이언트 측 임시 연결시 자동 할당.

따라서 실제 개발에서는 1024 이상의 포트를 사용하는 것이 안전하다.

예제 코드에서는 TEXT_PORT = 9999를 사용하였다.

QUdpSocket 초기화 및 바인딩

QUdpSocket 객체를 생성하고, 특정 IP와 포트에 바인딩하여 수신 대기 상태를 만든다.

```
text_socket = new QUdpSocket(this);

        if (text_socket->bind(OPERATOR_IP, TEXT_PORT,
                QUdpSocket::ShareAddress)) {connect(text_socket,
                SIGNAL(readyRead()), this, SLOT(udp_read()));}
```

bind(): 소켓을 IP와 포트에 연결.
readyRead 시그널: 데이터 수신 시 호출될 슬롯 함수(udp_read) 연결.

데이터 수신 (udp_read 함수)
QByteArray 버퍼를 선언하고, 수신된 데이터그램을 readDatagram()으로 읽는다.

```
void MainWindow::udp_read() {
    QByteArray buffer;
    buffer.resize(text_socket->pendingDatagramSize());
    text_socket->readDatagram(buffer.data(), buffer.size(),
&ROBOT_IP, &TEXT_PORT);
}
```

pendingDatagramSize(): 수신 대기 중인 데이터그램 크기 반환.

readDatagram(): 데이터, 송신자 IP, 포트를 함께 수신.

데이터 송신 (udp_write 함수)
ByteArray 형태의 데이터를 특정 IP와 포트로 송신한다.

```
void MainWindow::udp_write(QByteArray text, uint16_t port, QUdpSocket &socket) {
    QByteArray packet;
    packet.push_back(text);
    socket.writeDatagram(packet, ROBOT_IP, port);
}
```

writeDatagram(): 지정된 목적지 IP와 포트로 데이터 전송.
송신 측에서는 별도의 연결 설정 없이 즉시 전송 가능.

종합 분석

- 포트 관리: 실제 서비스 충돌을 피하기 위해 1024 이상의 포트 사용 권장.
- 데이터 수신: QByteArray 버퍼를 통해 유연하게 다양한 크기의 데이터 처리 가능.
- 데이터 송신: 단순한 API로 실시간 데이터 전송에 적합.
- 활용 분야: 로봇 영상 전송, 명령 제어 신호 송수신, 센서 데이터 스트리밍.

Qt의 QUdpSocket은 UDP 통신을 직관적이고 효율적으로 구현할 수 있게 한다. 포트 번호 체계에 따라 안전한 번호를 선택하고, bind(), readDatagram(), writeDatagram() 함수를 적절히 활용하면 안정적인 UDP 기반 응용 프로그램을 개발할 수 있다. 특히 실시간 데이터 교환이 중요한 로봇 제어 및 스트리밍 환경에서 효과적이다.

<실습>

1. 문제 요지

이번 과제는 카카오톡과 유사한 채팅 프로그램을 직접 구현하는 것을 목표로 한다. 단순히 메시지를 입력하고 출력하는 수준을 넘어서, 실제 메신저 서비스가 갖추어야 할 핵심 기능들을 모의 환경에서 재현하는 것이다. 이를 통해 네트워크 통신, UI 구성, 이벤트 처리, 데이터 저장 및 관리 등 다양한 프로그래밍 기술을 종합적으로 학습하는 데 목적이 있다. 학생은 메시지 송수신 과정과 사용자 인터페이스 흐름을 설계하고 구현하면서, 실무와 가까운 경험을 쌓게 된다.

2. 필수 조건

과제에서 반드시 구현해야 할 조건은 다음과 같다.

네트워크 기능: UDP 또는 TCP 소켓을 활용해 사용자가 입력한 메시지를 다른 사용자에게 전송하고, 동시에 수신된 메시지를 화면에 실시간으로 출력해야 한다.

UI 구현: 대화창, 입력창, 전송 버튼 등 기본 인터페이스를 Qt 같은 GUI 라이브러리로 구성하여 사용자가 직관적으로 이용할 수 있어야 한다.
다중 사용자 지원: 최소 2명 이상이 같은 네트워크 내에서 동시에 접속해 대화할 수 있도록 구조를 설계해야 한다.
메시지 기록: 송수신된 대화 내용을 로그 형태로 저장하거나 화면에 누적 표시하여, 이전 대화를 확인할 수 있어야 한다.
안정성: 연결 오류나 예외 상황 발생 시 프로그램이 즉시 종료되지 않고 예외 처리를 통해 복구할 수 있어야 한다.

3. 핵심 요약

이 과제의 핵심은 메신저의 기본 동작 원리를 직접 구현하는 것이다. 네트워크 통신을 기반으로 메시지를 교환하고, GUI를 통해 직관적인 대화 환경을 제공하며, 여러 사용자가 동시에 참여할 수 있는 구조를 완성해야 한다. 이를 통해 학생은 네트워크 프로그래밍, 이벤트 기반 UI 처리, 데이터 흐름 관리, 오류 처리 같은 실질적인 개발 역량을 종합적으로 훈련하게 된다.

1. mainWindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QHostAddress>
MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
, ui(new Ui::MainWindow)
, sock(new QUdpSocket(this))// QUdpSocket 객체 생성
{
    ui->setupUi(this);
// 내 PC의 IP와 포트를 바인드해서 수신 대기 시작
QHostAddress myAddr("172.100.4.199");// 내 PC의 사설 IP 주소
quint16 myPort =45454;// 내가 수신할 포트 번호
```

```
    sock->bind(myAddr, myPort,
                                        QUdpSocket::ShareAddress
|QUdpSocket::ReuseAddressHint);
// 데이터 수신 이벤트가 발생하면 onReadyRead() 슬롯 실행
            connect(sock,      &QUdpSocket::readyRead,      this,
&MainWindow::onReadyRead);
}
MainWindow::~MainWindow() { deleteui; }
void MainWindow::on_pushButton_clicked() {
// 내가 보낼 메시지 가져오기
QString msg =ui->send->toPlainText();
QByteArray dat =msg.toUtf8();
// 상대방의 IP와 포트 지정 (상대방은 이 포트로 바인드해야 수신 가
능)
QHostAddress peer("172.100.7.254");
    sock->writeDatagram(dat, peer, 45455);
// 내 화면(textEdit)에 내가 보낸 메시지도 바로 표시
    ui->textEdit->append("나: "+msg);
// 입력창 비우기
    ui->send->clear();
}
void MainWindow::onReadyRead() {
// 수신 버퍼에 대기 중인 데이터그램이 있으면 모두 처리
    while(sock->hasPendingDatagrams()) {
QByteArray buf;
        buf.resize(int(sock->pendingDatagramSize()));// 수신 데이
터 크기만큼 버퍼 준비
QHostAddress sender;
quint16 senderPort;
// 실제 데이터 읽어오기
            sock->readDatagram(buf.data(),  buf.size(),  &sender,
&senderPort);
// 수신한 메시지를 화면(textEdit)에 "상대:" 라벨과 함께 출력
        ui->textEdit->append("상대: "+QString::fromUtf8(buf));
}
}
```

이 코드는 Qt의 QUdpSocket을 활용한 간단한 카카오톡(메신저) 구

조를 보여준다. 전체 동작은 크게 (1) 소켓 초기화 및 수신 대기, (2) 버튼 클릭 시 메시지 송신, (3) 수신 이벤트 처리의 세 단계로 구성된다.

1. 소켓 초기화 및 수신 대기

생성자에서 QUdpSocket sock 객체를 초기화한다. 이어서 bind()를 통해 내 PC의 특정 IP(172.100.4.199)와 포트(45454)에 바인드하여, 해당 주소로 들어오는 데이터그램을 수신할 준비를 한다. ShareAddress와 ReuseAddressHint 옵션은 여러 프로그램이나 여러 소켓에서 같은 포트를 공유하거나 재사용할 수 있도록 허용하는 설정이다.

그 다음, Qt의 시그널-슬롯 메커니즘을 이용해 readyRead 시그널과 onReadyRead() 슬롯을 연결한다. 즉, 상대방이 메시지를 보내서 내 소켓에 데이터그램이 들어오면 자동으로 onReadyRead()가 호출된다.

2. 버튼 클릭 시 메시지 송신

on_pushButton_clicked() 함수는 사용자가 GUI에서 전송 버튼을 누를 때 실행된다.

먼저 입력창(ui->send)에 사용자가 작성한 텍스트를 가져와 QString msg로 저장한다.

이 문자열을 네트워크 전송이 가능한 QByteArray 형태로 변환한다.

이어서 상대방의 IP(172.100.7.254)와 포트(45455)를 지정한 뒤, writeDatagram()을 호출하여 메시지를 보낸다.

전송 즉시 내 대화창(ui->textEdit)에도 "나: 메시지" 형식으로 추가 표시하여, 사용자가 전송 내역을 확인할 수 있도록 한다.

마지막으로 입력창을 비워서 새로운 메시지를 바로 입력할 수 있게 한다.

3. 수신 이벤트 처리

상대방이 보낸 메시지는 onReadyRead() 함수에서 처리된다.
hasPendingDatagrams()로 현재 대기 중인 데이터그램이 있는지 확인한다.
데이터그램이 있으면 pendingDatagramSize()로 크기를 알아내고, 그만큼 버퍼(QByteArray buf)를 준비한다.
readDatagram()을 호출해 실제 데이터를 버퍼로 읽어온다. 이때 송신자의 IP와 포트도 함께 얻을 수 있다.
읽어온 메시지를 UTF-8 문자열로 변환해 "상대: 메시지" 형태로 내대화창(ui->textEdit)에 표시한다.
이 과정을 반복하여 수신 대기 중인 모든 데이터그램을 처리한다.

4. 정리
즉, 이 프로그램은 UDP 통신 기반 양방향 채팅을 구현한 예제다.
시작 시 내 소켓을 IP/포트에 바인드하여 수신 대기.
사용자가 버튼을 누르면 입력 메시지를 지정된 상대방 주소로 전송.
상대방이 보낸 메시지가 도착하면 자동으로 이벤트가 발생해 수신 처리 후 화면에 출력.
UDP 특성상 연결 설정 과정이 없고, 빠르지만 신뢰성(순서 보장, 손실 복구)은 TCP보다 낮다. 따라서 이 예제는 기본 메신저 구조 학습용으로 적합하다.

2. result screen

영상 참조 바람.

-끝-