HW2 보고서

로빛 20기 인턴 모시온

2025407006 로봇학부

목차

1. 개요

1-1. 보고서 개요

본 보고서는 실제 통신을 통한 하드웨어와의 정보 교환을 QT를 활용하여 시각화를 통해 본격적으로 하드웨어와 스프트웨어의 상호작용을 GUI로 사용자가 접근하여 적극적인 관여가 편리하게 작용할 수 있으며 그러한 통신의 종류 중 UDP를 통한 보고서 작성을 통해 개념과 응용을 이해하고자 합니다.

1-2. Qt와 네트워크 프로그래밍의 중요성

우선 기본적으로 Qt의 GUI 개발 툴은 다양한 운영체제에서 운영 가능한 높은 호환성을 지닙니다.
GUI의 설치된 각각의 모듈(버튼, 입력)에 대해 슬롯과 신호를 제공하며 이를 C++로 통한 접근이 가능합니다.

동시에, 현재의 소프트웨어 대부분은 공유된 클라우드를 통해
서로의 정보를 교환하는 것 뿐만이 아닌 정보를 관리하며 이를 통해 지정된 상호작용을 이루기 때문에 네트워크의 중요성 또한 비례하여 중요시되고 있습니다.

Qt와 네트워크의 결합을 지금 시점으로 보았을 때,
점점 중요시되어가며 단일 툴(또는 프로그램)에서 전체 기능을 구현하는 것이 아닌 툴과 툴간의 이식, 통신, 상호작용, 관리 및 제어를 통해 대규모의 프로젝트가 이루어지는 사회에서
오픈 소프트웨어로 제공되는 Qt를 통한 GUI 개발은 오픈 소프트웨어로서 서로간의 호환성이 매우 높으며, 이를 기반으로
다양한 용도 활용 및 언제든지 다른 플랫폼으로 이식할 수

있는 장점을 서로 다른 운영체제에서도 공통적으로 공유되는
통신(CAN, USB, UDP, 등등)에도 접목함으로서 전체적으로 계획되는 프로젝트의 유연함과 확장을 이룰 수 있다는 결과를 가져오게 됩니다.

1-3. UDP 통신의 활용 분야

(본 보고서는 해당 통신 중 UDP를 중점으로 분석합니다.)
간단하게, UDP란 비연결형 통신 활용으로서 상대방의 접속 여부에 구속받지 않고 자유롭게 데이터를 송신하는 구조로서 중간 데이터 점검 절차가 포함되어 있는 다른 통신과 달리 해당 과정을 생략함으로서 통신 속도를 비약적으로 상승시킬 수 있는 장점을 내포합니다.

이러한 장점을 내포하는 UDP은 현재 실시간 통신과 더불어 빠른 데이터 전달이 요구되는 프로젝트에 활용됩니다.
음성 신호를 통해 서비스를 제공하는 셋톱박스, 실시간 스트리밍 서비스와 같이 실시간 상호작용을 중심으로 활용되는데 이는 음성, 영상(수많은 사진 레이어의 재생)과 같이 연속적인 데이터 상으로는 일정 수준의 데이터 손실에 대해 연속적으로 받는 데이터를 통해 이를 보정할 수 있기 때문입니다.

음성 신호의 경우, 일반적으로 8kHz의 샘플링을 통해 연속적 데이터를 구현합니다. 이러한 데이터를 송수신 하는 경우에 발생하는 데이터 손실은 음성의 진동 수, 높낮이를 분석하여 복원이 가능한 방식을 통해 UDP 통신에도 무리가 없음을,
영상 신호의 경우, 적어도 24fps(1초에 24장 재생)의 영상 프레임을 보게 된다고 하였을 때, 그 중 1fps의 손실이 발생하거나 1fps 중 몇몇 픽셀이 깨지는 등의 데이터 변질 또는 손실이 발생하여도 바로 다음에 재생되는 사진 레이어를 통해 보정이 가능하여 UDP 통신이 적극 활용되고 있습니다.

(음성 신호 복원에 대한 근거입니다.)
https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE06091413#a
<강인한 음석 인식을 위한 알고리즘>

2. Qt 개요

2-1. Qt의 특징

Qt는 기본적으로 C++을 통해 사용자가 상호작용할 수 있는 GUI를
개발할 수 있는 오픈 소스 소프트웨어이며 이때, GUI뿐만이 아닌 네트워크 설계, 데이터베이스 등의 작업 또한 지원합니다.

사용자가 생성한 GUI 모듈(버튼, 입력)들을 C++를 통하여 객체 제어를 지원하는 독자적인 신호 및 슬롯을 제공합니다. 이에 추가적으로 위젯, 시간, 네트워크, 다량의 미디어 관리 서비스를 제공하는 모듈을 필요에 따라 선택적으로 적용할 수 있어 동일 툴 사용자에게 있어 서로의 코드 가독성이 다른 툴에 비해 높으며 안정성 및 확장성을 보장받게 됩니다.
위의 Qt의 특징은 자체 속성, 오픈 소스 소프트웨어의 특징에 따라 더 부각되며, C++를 활용할 시 C++의 객체지향형에 따른 장점 또한 내포되는 특징을 지닙니다.

2-2. Qt 주요 모듈 소개

QtCore

- Qt의 기반이 되는 모듈이며, 모든 Qt 애플리케이션에서 사용되는 이 모듈은 객체, 특정 이벤트, 데이터 구조, 시간 관리, 파일 입출력 및 신호와 슬롯 등의 기능을 처리합니다.
특히 Core는 GUI와 직접 관련된 기능을 제외한 순수한 비주얼

로직과 시스템 연동 기능을 제공함으로서, 콘솔 앱이나 로직 개발에서도 필수적인 역할을 합니다.

QtGui

- QtGui 모듈은 그래픽과 관련된 기능을 담당합니다. 2D, 사진, 글자, 색상, 픽셀 단위의 작업 처리와 관련된 Class를 제공하며, 화면에 표시되어 시각화 되는 객체의 표현을 담당합니다.

예를 들어, QPixmap이나 QImage를 사용해 사진 데이터를 불러오거나 변형이 가능하며, QPainter를 통해 직접 도형을 그리거나 텍스트를 출력할 수 있습니다.

QtNetwork

- QtNetwork 모듈은 네트워크 통신과 관련된 기능을 제공합니다. TCP, UDP 등 다양한 프로토콜 기반의 소켓 통신을 지원하며, 클라이언트와 서버 구조를 간편하게 관리할 수 있습니다.

특히 UDP 통신을 위한 QUdpSocket Class와 TCP 통신을 위한 QTcpSocket, QTcpServer Class는 비동기 이벤트 처리와 신호 및 슬롯 메커니즘을 통해 GUI와 통합된 실시간 통신을 구현할 수 있게 합니다. 또한, DNS 조회, SSL/TLS 보안 통신, 네트워크 인터페이스 관리 등 고급 기능도 제공하여, 현대 사회의 앱 개발 또는 UI 개발 등에서 요구되는 다양한 네트워크 요구사항을 충족할 수 있습니다.

2-3. 개발 환경 구성

Qt는 기본적으로 다음과 같은 환경 서비스를 제공합니다.

Qt Creator

-Qt Creator는 Qt 공식 통합 개발 환경 서비스이며 프로젝트 관리, 코딩, 디버깅, UI 디자인, 빌드 및 실행을 한 곳에서 수행할 수 있도록 제공되는 환경입니다.

Qt Creator는 Qt Designer를 내포하여 간단한 마우스의 드래그

와 같은 조작으로 GUI를 직관적으로 설계할 수 있으며, 생성된 UI 파일은 자동으로 C++ 코드와 연결됩니다.

qmake

-qmake는 Qt에서 제공하는 전통적인 빌드 관리 도구로, 프로젝트 파일에 포함된 설정을 기반으로 Makefile을 생성합니다. qmake는 소스 코드, 헤더 파일, 리소스 및 라이브러리 등을 관리하며, 각 운영체제별로 적합한 빌드 구조를 자동으로 적용하여 Qt 기본 모듈을 포함한 프로젝트에서 필요한 라이브러리와 헤더 설정을 자동화합니다.

CMake

-CMake를 이용한 빌드도 현재 사용됩니다.

CMake는 플랫폼 빌드 도구로서, 복잡한 프로젝트에서도 유연하게 빌드 설정을 관리합니다. CMake를 사용하면 Qt 프로젝트뿐만 아니라 일반 C++의 접근을 통한 외부 라이브러리 연동을 체계적으로 관리할 수 있으며, 다양한 컴파일러와 개발 환경에서 동일한 빌드 구성을 재현할 수 있습니다.

Qt 6 이후 버전에서는 CMake가 적극적으로 활용되며, qmake와 달리 현대적이고 확장성 높은 빌드 환경을 제공합니다.

3. 네트워크 통신 기초

3-1. OSI 7계층 개요

OSI 7계층이란 네트워크 통신을 이루는 과정을 총 7단계로 분할한 것을 뜻합니다. 해당 7계층은 아래와 같이 나누어 집니다.

응용 계층: 사용자가 접속할 수 있는 기능 동작

표현 계층: 전송되는 데이터 형태 결정

세션 계층: 데이터 통신의 논리 연결 관리

전송 계층: 지정 포트를 통해 전송 동작

네트워크 계층: 데이터 노드간의 경로 지정

데이터 링크 계층: 데이터의 흐름 및 오류 점검

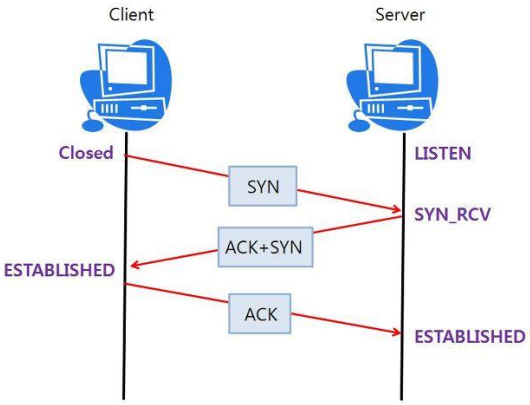
물리 계층: 전파를 통해 전달되는 데이터 전송

정리를 통해 서술하면, OSI 7계층은 데이터 전송 절차를 단계별로 분할하여 각 계층의 역할을 명확히 정의함으로서, 네트워크 설계와 문제 해결을 체계적으로 할 수 있도록 보조합니다.

3-2. TCP와 UDP 차이

TCP

-TCP란 전송 계층에서 동작하는 절차 요소로서 UDP와 달리 연결형 프로토콜 특징을 지닙니다. 이는 전송하는 데이터의 올바른 전송과정이 이루어졌는지를 데이터 전송 후 전송 대상에게서 받는 ACK(데이터 오류 판별 패킷)를 통해 오류의 여부를 확인 후 다음 데이터를 전송하는 방식을 뜻합니다.



왼쪽 사진의 경우

3way handshake TCP

데이터(SYN)

오류 판별 패킷(ACK)

이러한 TCP는 전달되는 데이터가 중요한 이메일, 파일 및 웹 서칭과 같은 서비스에 적극적으로 활용됩니다.

UDP

-반면 UDP는 비연결형 프로토콜로서, 전송 대상자의 올바른 데이터 수신에 관계없이 데이터를 데이터 그램 단위로 패킷을 전송합니다.

오류 판별 절차가 존재하지 않기 때문에 데이터의 손실에 따른 오류 및 변질이 발생할 수 있으나 절차 생략으로 통신 속도가 빠르다는 장점을 이전 목차에서도 언급했습니다.

3-4. 소켓 프로그래밍 개념

소켓 프로그래밍이란 데이터 송수신을 수행하는 프로그램을 사용하는 네트워크에 대해서 구현하는 기술입니다. TCP와 UDP와 같은 전송 계층 프로토콜과 결합되어 통신을 수행하게 되는데, TCP 소켓을 사용할 경우, 먼저 서버와 클라이언트 간 연결을 설정한 후, 연결이 확립되게 되었을 때 데이터를 송수신합니다. 이 과정에서 TCP는 본연의 기능으로서 오류 검사 및 데이터 송수신을 진행하며, 개발자는 소켓을 생성, 바인딩, 연결 및 데이터 전송등의 절차를 통해 안정적인 통신을 구현합니다.

UDP 소켓의 경우, 상대방의 연결과 무관하게 데이터를 송신, 데이터의 순서나 신뢰성을 보장하지 않지만, 절차가 간단하고 TCP 사용 시 데이터의 오류로 인해 재전송하는 데이터가 반복되어 추가적인 시간 또는 그외 자원이 소모되는 오버헤드가 있으나 USP는 적기 때문에 빠른 전송 속도를 제공합니다.

개발자는 바인딩 후 sendTo, readDatagram 등의 메서드를 통해 데이터를 송수신하여 통신을 구현하게 됩니다.

4. UDP 프로토콜 상세

4-1. 데이터 그램 구조

UDP의 데이터 송신 규격은 데이터그램에 따라 송신하게 되는데 해당 데이터그램은 다음과 같은 구조를 가집니다.

Header

-Header는 송수신 포트 채널, 길이, 체크섬 정보로 이루어지며, 포트 채널을 통해 데이터를 목적지와 연결하고, 체크섬을 통해 전송 중 발생할 수 있는 오류를 검출합니다. UDP 헤더는 8바이트로 고정되어 오버헤드가 적습니다.

Payload

-Payload는 실제 전송되는 사용자의 데이터로 구성되며, 데이터그램은 크기가 제한되어 있기 때문에 네트워크 MTU를 초과하면 전달하는 데이터가 분할되어 전송될 수 있습니다. 하지만 이 구조 덕분에 UDP는 송수신 과정이 단순해지며, 많은 데이터에도 빠른 전달이 가능하게 됩니다. 그러나 각 데이터그램은 독립적으로 전송되므로, 순서를 지정하지 않으면 손실된 패킷에 따른 보정이 어렵습니다.

4-2. 장점과 한계 (속도 vs 신뢰성)

전송 속도의 장점을 갖는 UDP에서 이러한 우려가 제기될 수도 있습니다. 연속적인 데이터에 특징에 따라 보정의 가능성을 두어 데이터 손실을 방치하는 양상을 보이게 됩니다.
하지만 어디까지나 위 UDP 통신 사용 시 감수해야하는 요소로서 사용자가 이를 숙지하고 데이터 손실에 크게 좌우되지 않는 통신을 담당함으로서 결과적으로 데이터 손실 자체를 큰 문제로 간주하지 않는 것입니다.

그럼에도, UDP의 한계는 신뢰성이 낮다는 것입니다.
데이터 손실, 순서 오류, 중복 수신 등이 발생할 수 있으며, 이러한 문제는 소프트웨어에서 직접 처리해야 합니다. 따라서 중요한 데이터 전송이나 파일 전송과 같이 데이터 완전성이 요구되는 경우에는 UDP보다 TCP가 더 적합하며 개발자 또한 이를 숙지하고 UDP에 정밀 데이터 수신을 의존하지 않도록 프로젝트 설계를 진행해야 합니다.
정리하면, UDP는 속도를 우선시하는 실시간 통신 환경에서 장점을 발휘하지만, 신뢰성이 필요한 환경에서는 한계가 존재합니다. 개발자는 구상 중인 프로젝트특성과 요구사항에 따라 UDP와 TCP 중 적절한 전송 방식을 선택해야 하며, 필요에 따라 UDP 위에서 독자적으로 설계한 오류 검출 및 재전송 로직을 구현하기도 합니다.

5. QtNetwork 모듈 소개

5-1. QTcpSocket, QUdpSocket 개요

QtNetwork모듈은 TCP와 UDP 기반의 소켓 통신을 쉽게 구현할 수 있는 클래스를 제공합니다.
대표적으로 QTcpSocket과 QUdpSocket이 있으며 각각 TCP와 UDP 프로토콜의 특성을 적용할 수 있습니다.

QTcpSocket은 TCP의 연결형 전송을 지원하여 서버와 클라이언트 간 데이터 송수신을 제공하며, 연결 설정 이후 송수신되는 데이터는 순서가 보장되어 오류가 발생할 시 자동으로 재전송이 수행하게 됩니다. 따라서 데이터 정밀성이 중요한 용도내에서 Qt Network모듈을 사용합니다.

QUdpSocket은 UDP의 비연결형 특성을 그대로 반영하여, 별도의

연결 없이 데이터그램 단위로 송수신이 이루어집니다. Qt를 통해 UDP를 사용하므로 UDP의 특징에 종속되어 사용되어집니다.

5-2. 신호/슬롯 기반 이벤트 처리 방식

Qt에서 소켓 통신에 발생되는 다양한 이벤트를 신호와 슬롯 메커니즘으로 처리됩니다.
그에 대한 예시로서, 데이터 수신 시 readyRead() 신호가 발생하게 되었을 때 이를 슬롯 함수와 연결한 후 자동으로 수신 데이터를 처리함으로서 구현됩니다.
이 방식은 이벤트 중심 구조를 최대한 단순화하며, GUI와 네트워크 통신을 자연스럽게 통합할 수 있다는 겁니다.
개발자는 별도의 함수 동작을 통해 반환되는 과정을 기다리는 지연없이, 이벤트가 발생할 때마다 인터럽트와 비슷한 방식으로 즉시 처리할 수 있습니다.
이러한 구조는 실시간 데이터 송수신과 GUI 업데이트가 동시에 필요한 앱에서 특히 유용하게 작용됩니다.

5-3. Qt에서 소켓 프로그래밍의 특징

Qt에서 소켓 프로그래밍은 다음과 같은 특징을 갖습니다.

비동기 통신

신호/슬롯 기반으로 이벤트 중심 통신을 구현하므로, 블로킹 없이 데이터를 송수신할 수 있습니다. 다양한 운영체제에서 동일한 코드로 소켓 통신을 구현될 수 있어 자체 이식성이 높습니다.

GUI와 네트워크

Qt에서는 GUI와 네트워크 모듈을 동시에 제공하기 때문에, 실시간으로 데이터와 화면 출력이 자연스럽게 융합됩니다.

간결한 코드 작성
QTcpSocket, QUdpSocket 클래스와 신호/슬롯 메커니즘을 활용하면, 이전의 C/C++ 에서 사용할 수 있던 소켓 코드보다 간결하게 네트워크 통신 코드를 작성할 수 있게 됩니다.

이러한 특징을 바탕으로 QtNetwork 모듈은 결과적으로 TCP와 UDP의 장점을 융합하여 살리면서 GUI 중심 프로젝트를 쉽게 결합 가능한 네트워크 환경을 제공합니다.

6. UDP 소켓 클래스(QUdpSocket)

6-1. 클래스 구조 및 주요 메서드

Qt에서 UDP 소켓 통신을 구현할 때 사용되는 QUdpSocket 클래스는 UDP 프로토콜의 비연결형 특성을 그대로 반영하며, 데이터그램 단위 송수신을 지원합니다. QUdpSocket은 QAbstractSocket을 상속받아, Qt의 신호/슬롯 기반 이벤트 처리와 통합된 구조를 갖습니다. 주요 메서드는 다음과 같습니다. bind(): 소켓을 특정 포트와 주소에 바인딩합니다. 수신용 소켓 설정에 필수적이며, 포트 충돌이나 접근 권한 문제를 체크합니다. writeDatagram() / sendTo(): UDP 데이터그램을 지정된 주소와 포트로 전송합니다. 연결 과정 없이 즉시 송신 가능하며, 빠른 전송 속도를 제공합니다. readDatagram() / pendingDatagramSize(): 수신된 데이터그램을 읽어 들이며, 데이터의 크기와 송신자 정보를 확인할 수 있습니다. 이 외에도 QUdpSocket은 소켓 상태 확인, 오류 처리, 로컬/원격 주소 확인 등 다양한 유틸리티 메서드를 제공하여 UDP 통신을 효율적으로 구현할 수 있습니다.

6-2. 바인딩, 송수신 과정

QUdpSocket을 이용한 UDP 통신은 바인딩과 데이터 송수신 과정으로 이루어집니다.

우선, 수신용 소켓을 bind() 메서드를 통해 로컬 포트와 주소에 연결합니다 이후 해당 포트로 들어오는 데이터그램을 수신할 준비를 하며 송신용 소켓은 별도의 연결 과정 없이 writeDatagram()을 통해 목적지와 포트로 데이터를 전송합니다.

UDP는 비연결형으로서 송신자는 수신자의 수신 여부를 확인하지 않고 데이터를 전송, 수신자는 들어오는 패킷을 독립적으로 사용자가 독자적으로 설계한 오류 패킷 처리과정이 있을 시 이를 처리합니다.

6-3.비동기 이벤트 처리

Qt에서 QUdpSocket은 신호/슬롯 기반의 비동기 이벤트 처리를 지원합니다.

이또한, 데이터가 수신될시 readyRead() 신호가 발생, 이후 슬롯 함수에서 즉시 데이터를 받아 처리하게 됩니다

이벤트 처리 기반의 구조로 인해 GUI에서도 메인코드가 정지되지 않고, 동시에 여러 데이터를 처리하면서 화면을 갱신할 수 있게 됩니다.

추가적으로, 여러 포트를 동시에 담당하거나, 송수신 이벤트를 동시에 처리하는 코드를 구성할 시 효율적이며. 결과적으로 QUdpSocket은 UDP의 장점을 유지하면서도, Qt의 이벤트 기반 프로그램과 결합되어 실시간 GUI 구현과 네트워크 통신을 동시에 구현할 수 있게 서비스를 제공합니다.

7. GUI와 UDP 연동

GUI 개발 위주로 유용하게 사용되는 Qt에서 위에서 알아본 GUI 및 UDP관련 QtNetwork 모듈을 통해 간단하게 송수신을 구현할 수 있는 프로그램 제작이 가능합니다.

7-1. 실시간 메시지 출력

실시간 메시지 출력은 UDP 수신 기능과 GUI 위젯의 결합을 통해 이루어집니다.

수신 데이터 바인딩

QUdpSocket Class로 만들어진 객체가 포트에 바인딩될 시, readyRead() 신호를 받아 들어오는 데이터가 이벤트를 발생하게 합니다.

출력 위젯 활용

QTextBrowser 또는 QListWidget을 사용하여 수신된 메시지를 즉시 화면에 출력할 수 있도록 합니다.

실시간성 보장

UDP는 ACK 확인 과정이 없기에, 데이터가 들어올때마다 GUI에 반영할 수 있으므로 채팅 시각화에 적합합니다.
이 방식을 통해 메시지가 제대로 수신되지 않아 누락되더라도 사용자에게는 데이터 전달을 지속적으로 확인할 수 있으며 따라서 다시한번, UDP의 장점이 원활하게 접목되어지는 것을 알 수 있습니다.

7-2. 사용자 입력 처리

사용자 입력 처리는 각각의 사전에 설치된 QLineEdit 또는 기타 입력란을 통해 지정된 문자열을 전송 기믹을 통해 QUdpSocket의 writeDatagram()이 동작하게 되어 지정 문자열이 해당 메서드로 전달되며 UDP 송신을 이룰 수 있습니다

8. 예시 적용

보고서 작성과 동시에 분석하게 된 Qt에서의 UDP 사용을 통해 제작한 과제1(hw1)에 대해서 설명합니다.

8-1. .h/.hpp 헤더 파일 설계

```
#include <QMainWindow>
#include <QUdpSocket>
#include <QTextEdit>
#include <QLineEdit>
#include <QPushButton>
#include <QHostAddress>
```

다음의 모듈을 호출하였습니다.

Qt의 위젯 활용과 사용자의 문자입력을 메서드로 전달할 QLineEdit 및 문자 기록을 남길 QTextEdit을 호출하며 UDP의 와이파이 연결을 위해서 QHostAddress를 불러왔습니다.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

이후 public 영역에서 Q_OBJECT를 통해 위젯 기능 활성화와 생성자 및 소멸자를 지정하여 초기화를 이루도록 하였습니다.

```
private slots:
    void sendMessage();
    void receiveMessage();
```

본 과제1은 문자의 송수신을 통한 채팅 기능을 구현하므로 2개의 각각 문자열 송신과 수신 함수를 private 슬롯으로 선언하며

```
private:
    QTextEdit *chatDisplay;
    QLineEdit *messageInput;
    QLineEdit *nickname;
    QLineEdit *target_address;
    QLineEdit *rx_port;
    QLineEdit *tx_port;
    QPushButton *sendButton;
    QUdpSocket *udpSocket;
```

private에서는 각각의 문자열을 저장할 멤버 변수들을 각각의 UI 모듈(입력란, 버튼 등등)에 따라 선언하였습니다.
역할은 아래와 같습니다.

*chatDisplay	-	대화 기록을 남기는 TextEdit 기능
*messageInput	-	입력(LineEdit)을 통해 메시지 저장
*nickname	-	입력(LineEdit)을 통해 닉네임 저장
*target_address	-	입력(LineEdit)을 통해 상대주소저장
*rx_port	-	입력(LineEdit)을 통해 수신포트저장
*tx_port	-	입력(LineEdit)을 통해 송신포트저장
*sendButton	-	버튼(PushButton)을 통해 문자전송

*udpSocket - UDP 통신으로 보낼 문자 저장

8-2. .c/.cpp 소스 파일 설계

```
//헤더파일 내의 멤버함수 생성 초기화
chatDisplay = new QTextEdit(this);
chatDisplay->setReadOnly(true);
```

```
messageInput = new QLineEdit(this);
sendButton = new QPushButton("Send", this);
target_address = new QLineEdit(this);
target_address->setPlaceholderText("Target IP");
rx_port = new QLineEdit(this);
rx_port->setPlaceholderText("Receive Port");
rx_port->setValidator(new QIntValidator(1, 65535, this));
nickname = new QLineEdit(this);
nickname->setPlaceholderText("Nickname");
```

이전에 지정한 각각의 입력란(QLineEdit)과 버튼(QPushButton)을 생성하여 초기화를 하였습니다.

```
QHBoxLayout *inputLayout = new QHBoxLayout();
inputLayout->addWidget(messageInput);
inputLayout->addWidget(sendButton);
inputLayout->addWidget(target_address);
inputLayout->addWidget(rx_port);
inputLayout->addWidget(tx_port);
inputLayout->addWidget(nickname);

layout->addWidget(new QLabel("Chat:"));
layout->addWidget(chatDisplay);
layout->addLayout(inputLayout);
```

2인 1조로 과제를 완성할 수 있게 하기 위하여 서로의 코드를 바로 이식할 수 있게 UI는 별도의 디자이너를 통한 편집이 아닌 코드로서 구현하여 이를 위해 위의 사진과 같이 위젯 추가 함수 addWidget을 통해 추가를, 이를 시각화하기 위하여 inputLayout 사용을 하였습니다.

```
    udpSocket = new QUdpSocket(this);
    udpSocket->bind(QHostAddress::Any, 8081, QUdpSocket::ShareAddress |
QUdpSocket::ReuseAddressHint);
```

UDP 통신 활용을 위해 생성과 바인딩을 하였습니다.

```
connect(sendButton, &QPushButton::clicked, this, &MainWindow::sendMessage);
connect(messageInput, &QLineEdit::returnPressed, this, &MainWindow::sendMessage);
connect(udpSocket, &QUdpSocket::readyRead, this, &MainWindow::receiveMessage);
```

생성한 위젯들을 각각의 함수에 연결하기 위하여 connect를 활용해 연결하였으며,

```
    connect(rx_port, &QLineEdit::editingFinished, this, [=](){
        quint16 port = rx_port->text().toUShort();
        if(port > 0) {
            udpSocket->close();
            udpSocket->bind(QHostAddress::Any, port, QUdpSocket::ShareAddress |
QUdpSocket::ReuseAddressHint);
            chatDisplay->append("Receive port changed to: " + QString::number(port));
        }
    });
}
```

위 코드는 사용자가 임의로 지정할 수 있는 송수신 포트를 각각의 rx_port와 tx_port를 활용하여 포트 변경을 이룰 수 있도록 한 부분입니다.
해당 포트값이 올바르지 않은 경우에는 닫히도록 예외처리하여으며 현재 본인이 설정한 포트 값을 보다 더 나은 시각화를 이룰 수 있도록 chatDisplay를 통해 안내 메시지를 출력하도록 하였습니다.

```
QString text = messageInput->text();
if (text.isEmpty()) return;

QString ip = target_address->text();
quint16 port = tx_port->text().toUShort();
QString name = nickname->text();
```

입력되는 문자열을 messageInput에 저장할 수 있도록 하며, 각각의 지정 포트 및 인터넷 주소를 적용하도록 하였습니다.

```
QString fullMessage = name + ": " + text;

QByteArray data = fullMessage.toUtf8();
udpSocket->writeDatagram(data, QHostAddress(ip), port);
```

그 다음 전체 송신 메시지를 fullmessage에 닉네임과 문자를 같이 저장하여 이를 송신하도록 하였습니다.

```
    while (udpSocket->hasPendingDatagrams()) {
        QByteArray datagram;
        datagram.resize(int(udpSocket->pendingDatagramSize()));
        QHostAddress sender;
        quint16 senderPort;

        udpSocket->readDatagram(datagram.data(), datagram.size(), &sender,
&senderPort);
        QString msg = QString::fromUtf8(datagram);
        chatDisplay->append(msg);
    }
```

해당 코드는 문자 수신과 관련된 코드입니다.
datagram이라는 배열을 선언하며 수신되는 문자에 따라 길이를 할당 및 해당 수신 메시지를 chatDisplay를 통해 문자 기록을 남길 수 있도록 하였습니다.

9. 프로젝트 확장 방안

9-1. TCP + UDP 하이브리드 모델

서로 대조되는 상반된 개념으로서 존재하는 TCP, UDP 통신은 분류되는 종류이나 실제 사용에서는 두 통신은 융합하여 하나의 하이브리드로서 교차 병행 사용을 통해 준수한 속도와 정밀도를 확보할 수 있으리라 생각됩니다.

실제적으로 완전히 두 통신을 결합할 수는 없으나 하나의 하드웨어에서 각각의 지정된 역할을 두 통신으로 나누어 송신하며 수신 장치에서 두 통신을 동시에 받을 수 있게 하여 효율성을 증가시킬 수 있는 가설입니다.

실제로 이러한 특징을 구현하기 위한 통신 중 SCTP 통신이 둘의 특성을 가장 잘 융합한 모델이라 예상됩니다.

SCTP (Stream Control Transmission Protocol)
위 통신은 TCP와 UDP의 장점을 서로 결합한 전송 계층 프로토콜로서 TCP처럼 연결 지향적이며 신뢰성을 보장합니다. 그럼에도 불구하고 동시에 UDP처럼 멀티스트리밍과 멀티호밍을 지원하여, 하나의 통신에서 여러 데이터를 병렬로 전송하여 최대한 지연을 줄이는 원리를 적용하여 설계된 모델입니다.