Obligatorisk oppgave 2

DFON2110 Operativsystemer, nettverk og programmeringsspråk

Sirajuddin Asjad (216988)

Innholdsfortegnelse

1 Introduksjon

1.1 Oppgavebeskrivelse

Denne oppgaven går ut på å lage en TCP klient som skal kommunisere med en TCP server på en gitt IP addresse og port. Kommunikasjonen foregår ved bruk av en spesifikk protokoll med noen spesifikke krav som må oppfylles, slik at bruker av klienten får kommunisert riktig med serveren. Dersom man har koblet seg til serveren og oppfyller alle krav til protokollen, så skal man holde en aktiv forbindelse med serveren ved at serveren pinger klienten og klienten må svare ved å ponge tilbake. På denne måten kan vi kommunisere med en server inntil enten klienten eller serveren stopper koblingen.

1.2 Valg av operativsystem og språk

TCP klienten har blitt programmert og utviklet i C++ med Qt Framework, og platformen som har blitt brukt er Linux. Klienten er programmert som GUI, altså programmet kjøres som et visuelt program og er ikke kommando-basert. Den forrige obligatoriske oppgaven ble programmert i Java, så derfor har utvikleren valgt å programmere denne nye obligatoriske oppgaven i C++, og for å gjøre oppgaven ekstra utfordrende og interessant så er den utviklet med GUI. Det som er utfordrende med GUI utvikling er at man må ta hensyn til mye annet enn hvis man kun kjører programmet i et kommando-basert grensesnitt, som for eksempel kunne håndtere tråder på riktig måte og kunne programmere vinduet godt nok til at det ikke krasjer eller trenger å restarte. Og ikke minst huske å rydde opp etter seg i minnet, siden slike GUI programmer bruker en del minne og dermed må dette tømmes for å ikke føre til minnelekkasje.

2 Implementasjon

2.1 Beskrivelse av kode

Siden programmet er bygd med Qt så er det enkelte filer og funksjoner som er definert på forhånd, altså disse må være her for at koden skal fungere. Dette gjelder for eksempel *TCPClient.pro* og *TCPClient.pro.user*. Selve designet til programmet ligger i *TCPClient.ui* filen, og dette blir generert når jeg bruker Qt Designer og eksporterer designet til en lesbar fil.
Koden starter selvfølgelig med en main funksjon, hvor vi initialiserer objektet vårt og forteller Qt at vi ønsker å vise vinduet vårt.

```
#include <QApplication>
#include "client.h"

int main(int argc, char *argv[]){
   QApplication a(argc, argv);
   TCPClient client;
   client.show();

   return a.exec();
}
```

Programmet har to header-filer, hvor vi har to forskjellige klasser. Hovedklassen som kjører selve Qt-vinduet er TCPClient, mens Socket-klassen blir brukt inne i TCPClient-klassen. I TCPClient-klassen er alle funksjonene relatert til klienten definert, mens i Socket-klassen er alle socket- og nettverksfunksjoner definert.

```
class TCPClient : public QMainWindow{
   Q_OBJECT

public:
   explicit TCPClient(QWidget *parent = nullptr);
   ~TCPClient();
   void addLog(QString);

private slots:
   void on_connectButton_clicked();
   void on_clearButton_clicked();

private:
   Ui::TCPClient *ui;
   Socket socket;
};
```

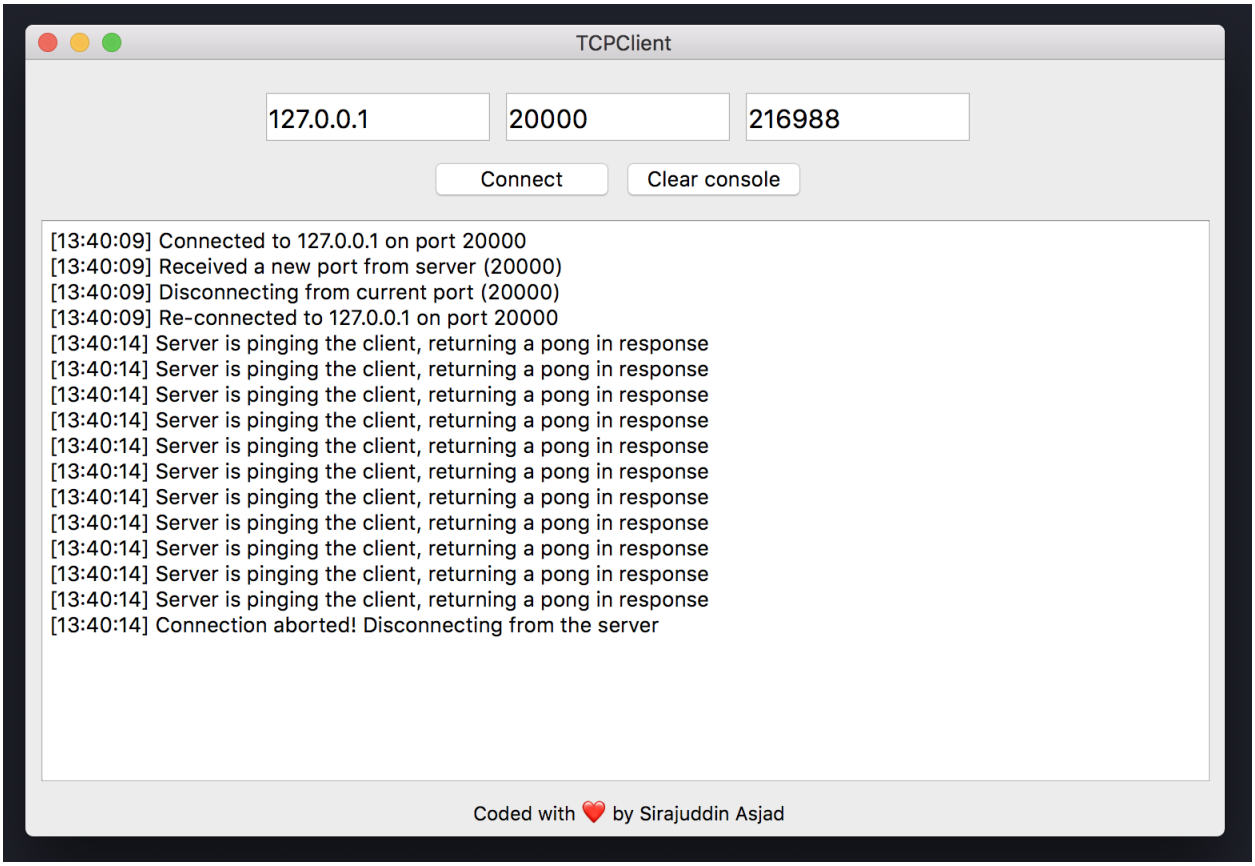
Det blir opprettet et objekt av Socket-klassen som er privat innenfor TCPClient-klassen, slik at vi får kjørt nettverksfunksjonene våre som ligger i Socket-klassen.

```
namespace Server{
   struct serverInfo{
      int sock, port;
      std::string ip;
      bool isConnected;
      sockaddr_in serv_addr;
   };

   struct studentInfo{
      char* number = new char[6];
   };

   enum MessageID{
      REQUEST_PORT = 0x01,
      RECEIVE_PORT = 0x02,
      PING = 0x03,
      PONG = 0x04,
      QUIT = 0x05
   };

   enum Errors{
      SOCKET_ERROR,
      INVALID_CONNECTION,
      INVALID_STUDNR,
      INVALID_PORTREQUEST,
      INVALID_PORTRESPONSE,
      PING_ERROR,
      PONG_ERROR
   };
};

class Socket{
public:
   Socket() : server{ 0, 0, "NULL", false } {}
   void makeConnection(std::string, int);
   void abortConnection();
   bool getConnectionStatus() const{ return server.isConnected; }
   char* getStudentNumber() const{ return student.number; }
   void verifyStudent(std::string);
   void requestPort();
   short receivePort();
   Server::MessageID portResponse();
   void pongServer();

private:
   Server::serverInfo server;
   Server::studentInfo student;
};
```

Alle funksjoner relatert til klienten ligger i *client.cpp* filen, mens alle funksjoner relatert til socket ligger i *socket.cpp*, slik at funksjonene er ryddig og lett å finne frem. I løpet av hele koden kjøres 7 funksjoner: makeConnection(), abortConnection(), verifyStudent(), requestPort(), receivePort(), portResponse(), pongServer().

2.2 Demonstrasjon

I figuren under er klienten koblet til serveren med riktig IP addresse, port og student nummer. Serveren vil da returnere en ny port som klienten må koble til, og deretter skal klienten holde en aktiv forbindelse ved å motta en ping og svare med pong inntil serveren sier stopp. Klienten er testet mot egen server som er programmert i Python med samme protokoller som Christians server.



3 Oppsummering

3.1 Konklusjon

Denne oppgaven har vært ekstremt lærerikt, og det har vært en veldig gøy opplevelse å kode en slik klient, med tanke på at fagstoffet er nytt og ukjent. I tillegg så er det øving i rapportskriving og LaTeX, og det er jo veldig gøy. Det er selvfølgelig alltid morro å kode i C++ og Qt med tanke på at det er utrolig mye man kan få til. Det har vært litt problemer og misforståelser med leveringsfrister og justeringer underveis, og det syntes jeg skulle vært bedre planlagt på forhånd. Men jeg har ingenting å klage på - jeg har kost meg masse og lært en del nye ting, og det setter jeg pris på.
Jeg har brukt mye tid på koden med tanke på at den er laget med GUI, og derfor brukt mindre tid på rapporten. Jeg skulle kanskje starte mye tidligere på rapporten, for da hadde jeg rukket å skrive en ordentlig og fin rapport. Ellers er dette en perfekt oppgave og det har vært en god opplevelse å utvikle noe slikt.

3.2 Referanser

[1] Christian Scott, Universitetet i Sørøst-Norge
Avdelingsingeniør, lærer og guru ved USN Kongsberg

[2] TCP/IP Sockets in C: Practical Guide for Programmers
Elsevier Science, 2009. Skrevet av Michael Donahoo, Kenneth Calvert

3.3 Kildekode

main.cpp:

```
#include <QApplication>
#include "client.h"

int main(int argc, char *argv[]){
   QApplication a(argc, argv);
   TCPClient client;
   client.show();

   return a.exec();
}
```

client.h:

```
#ifndef CLIENT_H
#define CLIENT_H
#include <QMainWindow>
#include <QDateTime>
#include "socket.h"

namespace Ui{
   class TCPClient;
}

class TCPClient : public QMainWindow{
   Q_OBJECT

public:
   explicit TCPClient(QWidget *parent = nullptr);
   ~TCPClient();
   void addLog(QString);

private slots:
   void on_connectButton_clicked();
   void on_clearButton_clicked();

private:
   Ui::TCPClient *ui;
   Socket socket;
};

#endif
```

socket.h:

```
#ifndef SOCKET_H
#define SOCKET_H
#include <iostream>
#include <string>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>

namespace Server{
   struct serverInfo{
      int sock, port;
      std::string ip;
      bool isConnected;
      sockaddr_in serv_addr;
   };

   struct studentInfo{
      char* number = new char[6];
   };

   enum MessageID{
      REQUEST_PORT = 0x01,
      RECEIVE_PORT = 0x02,
      PING = 0x03,
      PONG = 0x04,
      QUIT = 0x05
   };

   enum Errors{
      SOCKET_ERROR,
      INVALID_CONNECTION,
      INVALID_STUDNR,
      INVALID_PORTREQUEST,
      INVALID_PORTRESPONSE,
      PING_ERROR,
      PONG_ERROR
   };
};

class Socket{
public:
   Socket() : server{ 0, 0, "NULL", false } {}
   void makeConnection(std::string, int);
   void abortConnection();
   bool getConnectionStatus() const{ return server.isConnected; }
   char* getStudentNumber() const{ return student.number; }
   void verifyStudent(std::string);
   void requestPort();
   short receivePort();
   Server::MessageID portResponse();
   void pongServer();

private:
```

```
    Server::serverInfo server;
    Server::studentInfo student;
};

#endif
```

client.cpp:

```
#include "client.h"
#include "ui_client.h"

TCPClient::TCPClient(QWidget *parent) : QMainWindow(parent), ui(new Ui::TCPClient){
    ui->setupUi(this);
}

TCPClient::~TCPClient(){
    delete ui;
}

void TCPClient::addLog(QString text){
    QDateTime dateTime = dateTime.currentDateTime();
    QString dateTimeString = dateTime.toString("hh:mm:ss");
    ui->logBox->appendPlainText("[" + dateTimeString + "] " + text);
}

void TCPClient::on_connectButton_clicked(){
    if(!ui->ipBox->text().isEmpty() && !ui->portBox->text().isEmpty() &&
        !ui->studBox->text().isEmpty()){
        std::string tempIP = ui->ipBox->text().toStdString();
        int tempPort = ui->portBox->text().toInt();
        std::string studNumber = ui->studBox->text().toStdString();

        if(!socket.getConnectionStatus()){
            try{
                socket.verifyStudent(studNumber);
                socket.makeConnection(tempIP, tempPort);
                addLog("Connected to " + ui->ipBox->text() + " on port " + ui->portBox->text());

                socket.requestPort();
                short newPort = socket.receivePort();
                addLog("Received a new port from server (" + QString::number(newPort) + ")");
                addLog("Disconnecting from current port (" + ui->portBox->text() + ")");
                socket.abortConnection();

                socket.makeConnection(tempIP, newPort);
                addLog("Re-connected to " + ui->ipBox->text() + " on port " +
                    QString::number(newPort));
                ui->connectButton->setText("Disconnect");

                Server::MessageID response = socket.portResponse();
                while(response == Server::MessageID::PING){
                    addLog("Server is pinging the client, returning a pong in response");
                    socket.pongServer();
                    response = socket.portResponse();
                }
                if(response == Server::MessageID::QUIT) addLog("Connection aborted! Disconnecting
```

```
                from the server");
            socket.abortConnection();
            ui->connectButton->setText("Connect");
        }
        catch(Server::Errors e){
            switch(e){
                case Server::INVALID_STUDNR: addLog("Invalid student number!"); break;
                case Server::SOCKET_ERROR: addLog("Failed to create a socket!"); break;
                case Server::INVALID_CONNECTION: addLog("Failed to connect to the server!"); break;
                case Server::INVALID_PORTREQUEST: addLog("Failed to receive a new port from the
                    server!"); break;
                case Server::PING_ERROR: addLog("Server could not ping the client properly!");
                    break;
                case Server::PONG_ERROR: addLog("Client could not pong the server properly!");
                    break;
                case Server::INVALID_PORTRESPONSE: addLog("Invalid port response!"); break;
            }
        }
        catch(...){ addLog("Unknown error occured!"); }
    }
    else{
        socket.abortConnection();
        addLog("Disconnected from the server");
        ui->connectButton->setText("Connect");
    }
  }
  else addLog("Please enter the server IP address, port number and student number!");
}

void TCPClient::on_clearButton_clicked(){
  ui->logBox->clear();
}
```

socket.cpp:

```
#include "socket.h"

void Socket::makeConnection(std::string ip, int port){
   if(ip == "lekeplass") server.ip = "158.36.70.56"; else server.ip = ip;
   server.port = port;
   server.sock = socket(AF_INET, SOCK_STREAM, 0);
   if(server.sock == 0) throw Server::SOCKET_ERROR;

   server.serv_addr.sin_addr.s_addr = inet_addr(server.ip.c_str());
   server.serv_addr.sin_family = AF_INET;
   server.serv_addr.sin_port = htons(server.port);
   int c = connect(server.sock, (sockaddr*)& server.serv_addr, sizeof(server.serv_addr));
   if(c < 0) throw Server::INVALID_CONNECTION;
   server.isConnected = true;
}

void Socket::abortConnection(){
   close(server.sock);
   server.sock = 0;
   server.port = 0;
   server.ip = "NULL";
   server.isConnected = false;
}

void Socket::verifyStudent(std::string studNumber){
   if(studNumber.length() != 6) throw Server::INVALID_STUDNR;
   student.number = (char*)studNumber.c_str();
}

void Socket::requestPort(){
   char buffer[10];
   char messageID = Server::MessageID::REQUEST_PORT;
   const short packageSize = sizeof(packageSize) + sizeof(messageID) + strlen(student.number);

   memset(buffer, 0, sizeof(buffer));
   memcpy(buffer, &packageSize, sizeof(packageSize));
   memcpy(buffer+2, &messageID, sizeof(messageID));
   memcpy(buffer+3, student.number, strlen(student.number));

   int s = send(server.sock, buffer, sizeof(buffer), 0);
   if(s < 0) throw Server::INVALID_PORTREQUEST;
}

short Socket::receivePort(){
   char buffer[10];
   memset(buffer, 0, sizeof(buffer));
   int r = recv(server.sock, buffer, sizeof(buffer), 0);
   if(r < 0) throw Server::INVALID_PORTREQUEST;

   short newPort;
   memcpy(&newPort, buffer+3, sizeof(newPort));
   return ntohs(newPort);
}
```

```
Server::MessageID Socket::portResponse(){
   char buffer[10];
   memset(buffer, 0, sizeof(buffer));
   int r = recv(server.sock, buffer, sizeof(buffer), 0);
   if(r < 0) throw Server::PING_ERROR;

   switch(buffer[2]){
      case Server::MessageID::PING:{
         return Server::MessageID::PING;
      }
      case Server::MessageID::PONG:{
         return Server::MessageID::PONG;
      }
      case Server::MessageID::QUIT:{
         return Server::MessageID::QUIT;
      }
      default:{
         throw Server::INVALID_PORTRESPONSE;
      }
   }
}

void Socket::pongServer(){
   char buffer[10];
   char messageID = Server::MessageID::PONG;
   const short packageSize = sizeof(packageSize) + sizeof(messageID);

   memset(buffer, 0, sizeof(buffer));
   memcpy(buffer, &packageSize, sizeof(packageSize));
   memcpy(buffer+2, &messageID, sizeof(messageID));

   int s = send(server.sock, buffer, sizeof(buffer), 0);
   if(s < 0) throw Server::PONG_ERROR;
}
```