Universitetet i Sørøst-Norge

Fakultet for Teknologi, Naturvitenskap og Maritime fag

Institutt for realfag og industrisystemer

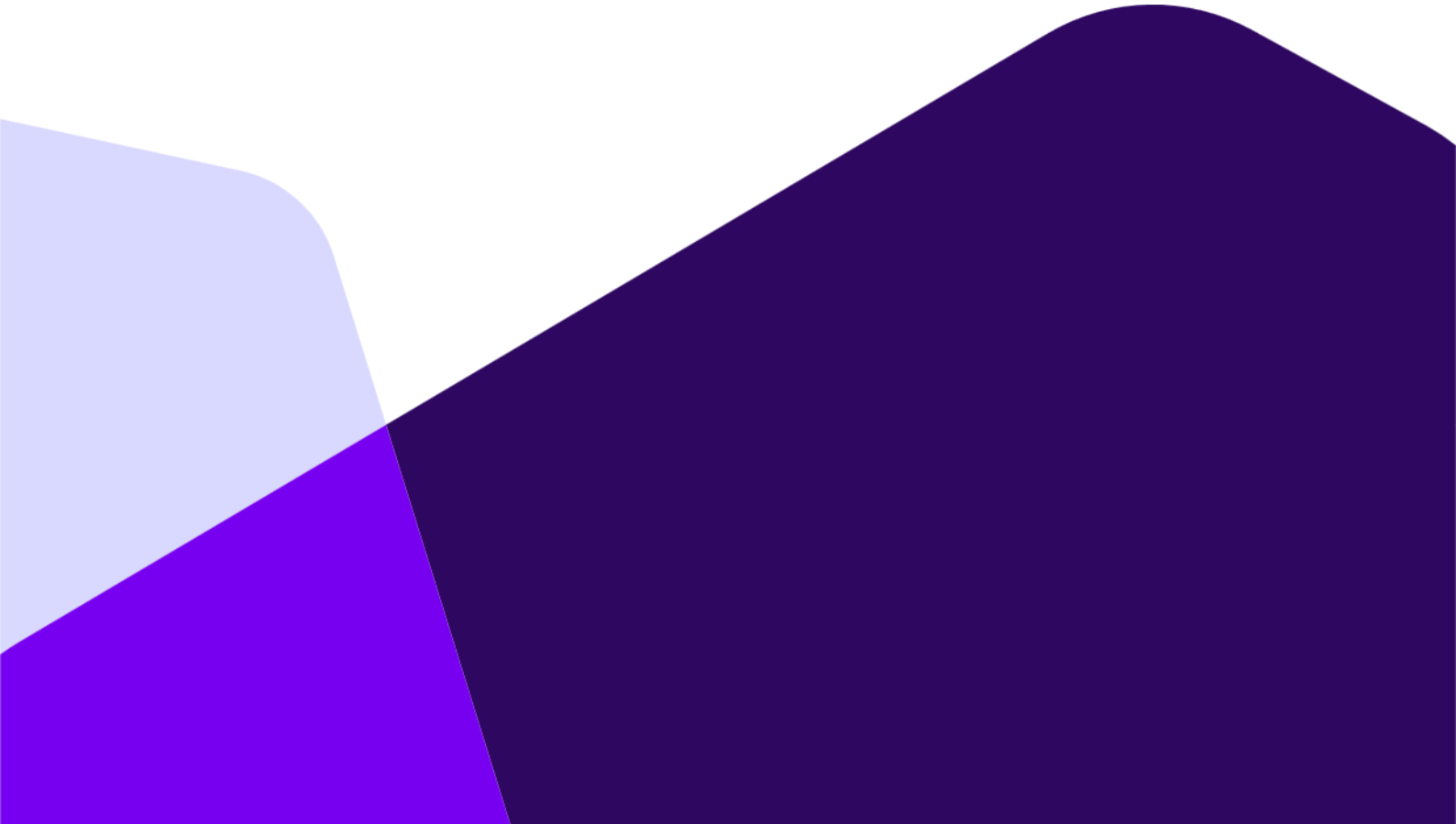
Innleveringsoppgave

Nettverk og sikkerhet/TSD2090K-1

24.02.2025

[Salim Zakaria Ahmed/241775]

Enkel melding/chat system



Om bruk av KI-verktøy

AI er et veldig kraftig middel. Jeg har ikke brukt dette mye siden det ikke er lærerikt. Jeg har søkt på nettet der jeg har fått hjelp med å skrive 1024+ bytes med alfabetet for å se hvor feilen hadde kommet. Lærte meg cin.ignore siden det var for mye i bufferet og fikk ikke noe ut i cin.getline(). jeg lærte også fra nettet at Uten byte='\0', kaster søppel som du ser på eksempel senere.

Kunne gjerne brukte det på å skrive feil til neste gang, og gjøre det mer systematisk. Middelet som hadde vært brukt er chatgpt. Har testen koden min der og den sier det er flere feil, selv om den kjører fint og besvarer oppgavens krav. Jeg brukte også chat med hjelp av riktig bibliotek. Jeg så fra nettet hvordan man håndterer ios::app i en ofstream som jeg har brukt i loggen.

Innholdsfortegnelse

Innledning

I denne oppgaven skal jeg ta for meg og lage en TCP/IPP chat system. Jeg skal implementere dette i et c++ program der den skal også koble til med andre og seg selv. Jeg skal lage en server og klient.

Min server skal kunne ha en enkel oppsett med formel WSAStartup som skal starte den. Så lager jeg socket. Deretter skal man binde socket. Så kommer høre på socket med listen. Deretter akseptere tilkoblingen med accept. Så skal det kunne sende og motta meldinger som fører til chatsystemet. Til slutt skal den koble seg fra.

Min klient har en lik start. Den skal først starte socket klienten med WSAStartup. Etterfulgt med å lage socketen. Men da skal den koble seg til en server med connect. Deretter kan man integrere med systemet med send og receive.

Hovedkapitlene

Jeg har satt opp koden slik at den er if statements. Hvis den er feil så lukker jeg og lukker socket. Ellers så går koden. Fleste av de som er satt som int som jeg lærte i øvingstimen returnerer en 1. jeg har en while-løkke med break for å komme ut hvis det er feil i recv(). Jeg setter inn manuelt port og ip adresse, dette med en int og en string. En i klient og en i server. Jeg lagde også in_addr og struct_in_adddr, men fikk ikke brukt dem siden inet_pton allerede er i biblioteket. Hadde det det for god læring.

Det er et par krav jeg må fylle for både server og klient. Server og klient skal kunne koble til andre dette har jeg gjort ved å bruke IPv4 AF_INET og port 8080, som man ser i koden kunne det vært flere muligheter. Og jeg brukte tinet_pton siden inet_addr ikke gikk. Jeg skrev den opp for god

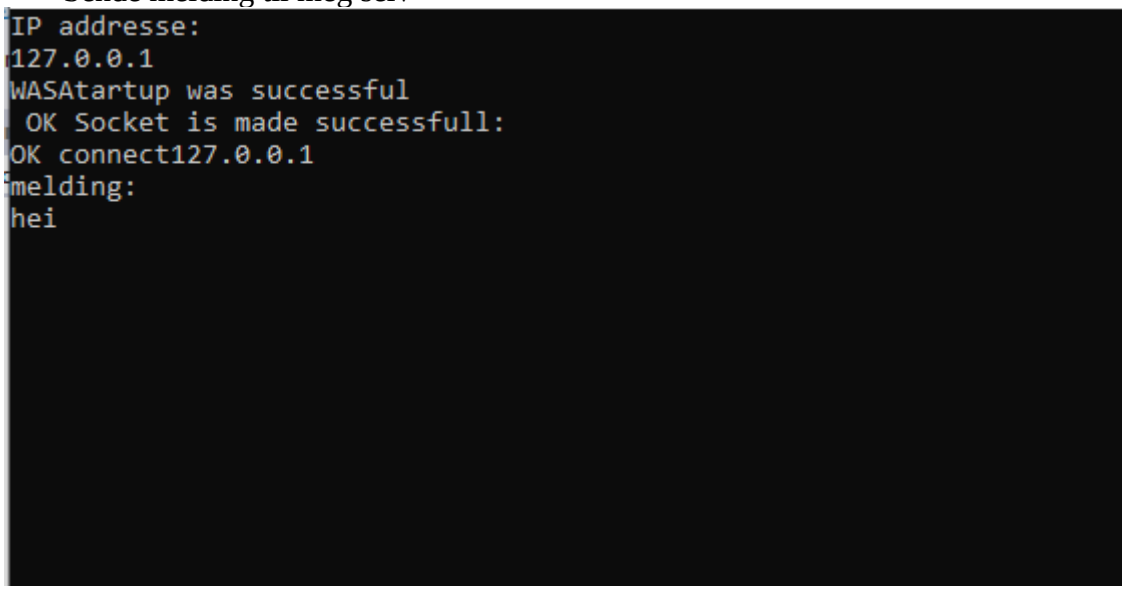
læring og starten har jeg konstruert dens addr som også ikke trengs siden den allerde er i biblioteket. Jeg satte inn logg på begge etter vært kall for å ha oversik. Dette brukte jeg en ofstream.til .Jeg brukte SIGTERM og SIGINT ved å håndtere ctrl+c for å exitet. Som vi lærete på timen. Bufferet skal være på 1024 bytes som jeg har gjort ved å kalle den buf. Klienten kan koble seg til andres servere ved å skrive inn IP adressen, men det må være under samme port og nettverk. Jeg har en enkel brukegrensesnitt ved å ha en cin.getline som tar inn teksten, og får ut ulike bytes. Jeg har nettverksfeil med socketError(). Dette er en unødvendig funksjon, men det er mer ryddig. Jeg har en ryddig avslutning med en melding, closeSocket() og cleanSocket().

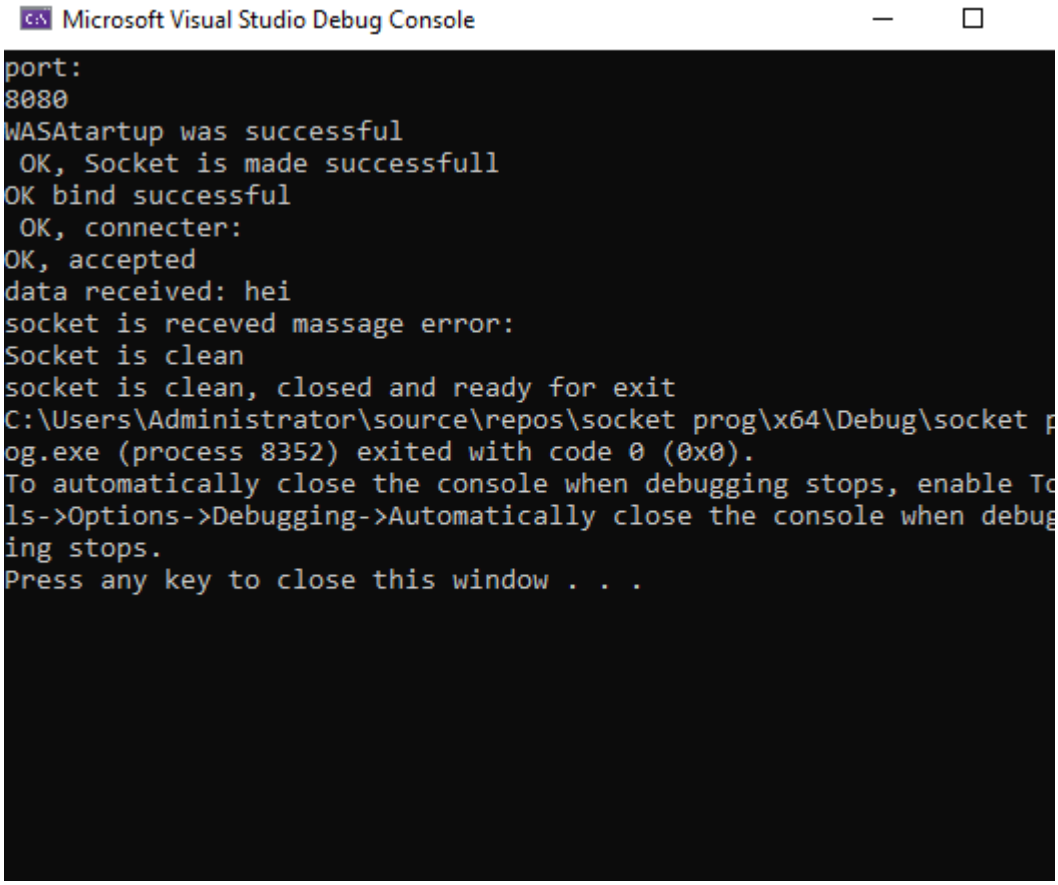
Både server og klient har robuste feilhåndteringer. Netteverksfeil er satt opp med SOCKET_ERROR og INVALID_SOCKET. Som hvis når tilkoblingen med nettet er brutt. Den håndterer også for ugyldig input så kommer ikke meldingen når du skriver inn feil IP-adresse. Et uventet brudd håndteres med buf[recvSocket] = '\0' som fjerner uventet data når systemet feilet. Som man ser i testingen. Bufferoverflow er håndtert med buf der det er 1024 bytes kun.
Disse feilhåndteringene kunne vært bedre.

Nedenfor kan du se test av feilmelidng og buffoverflow. Testeing av å kjøre med egen server, brukte 127.0.0.1 og sender hei. En melding som har mer enn 1024 bytes. Som man ser kjører koden, men stopper etter 1024. UTF-8 går, som man ser skrev jeg melding som «lære å øve», som kjører normalt. Et krav var også at 2 klienter skal koble seg til min server i IRL-Labben. Jeg gjorde det med 2 ulike perspektiver. Jeg tok også med ett eksempel der min klient kobler seg til en annen server.

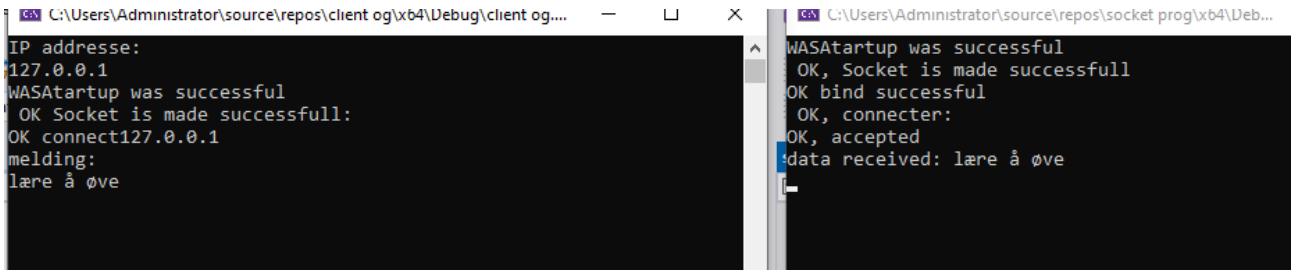
Test:

Sende melding til meg selv

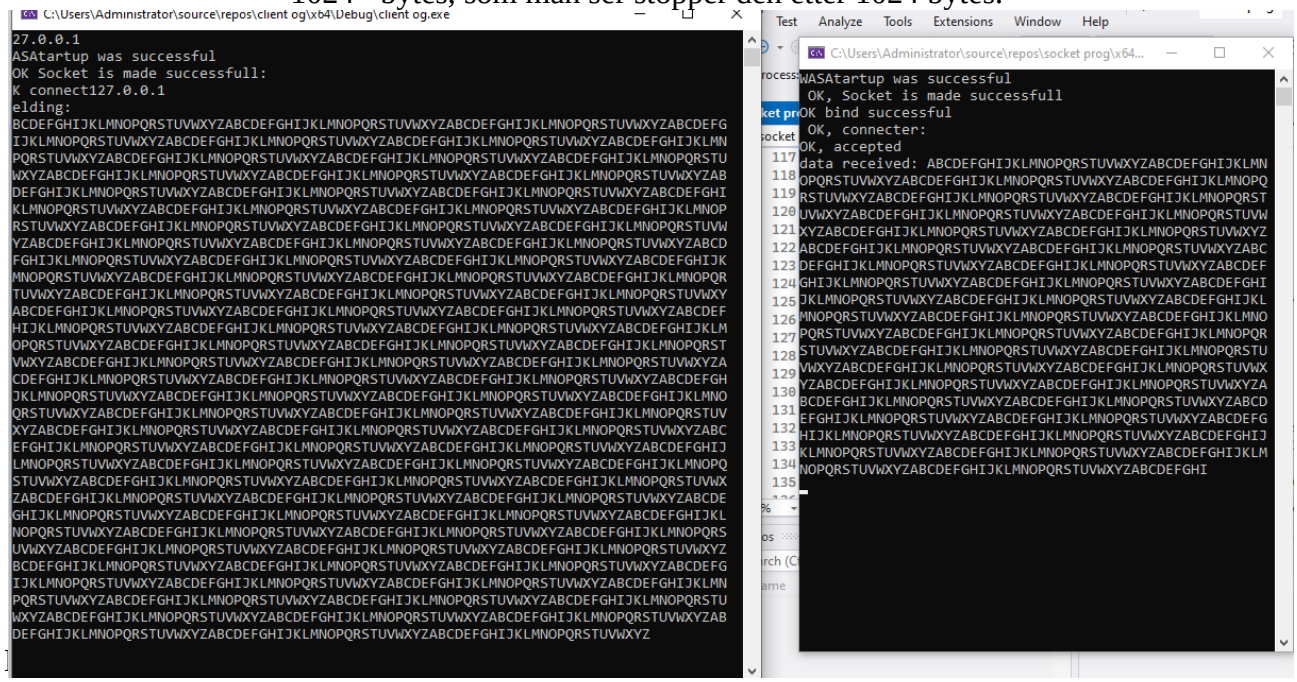




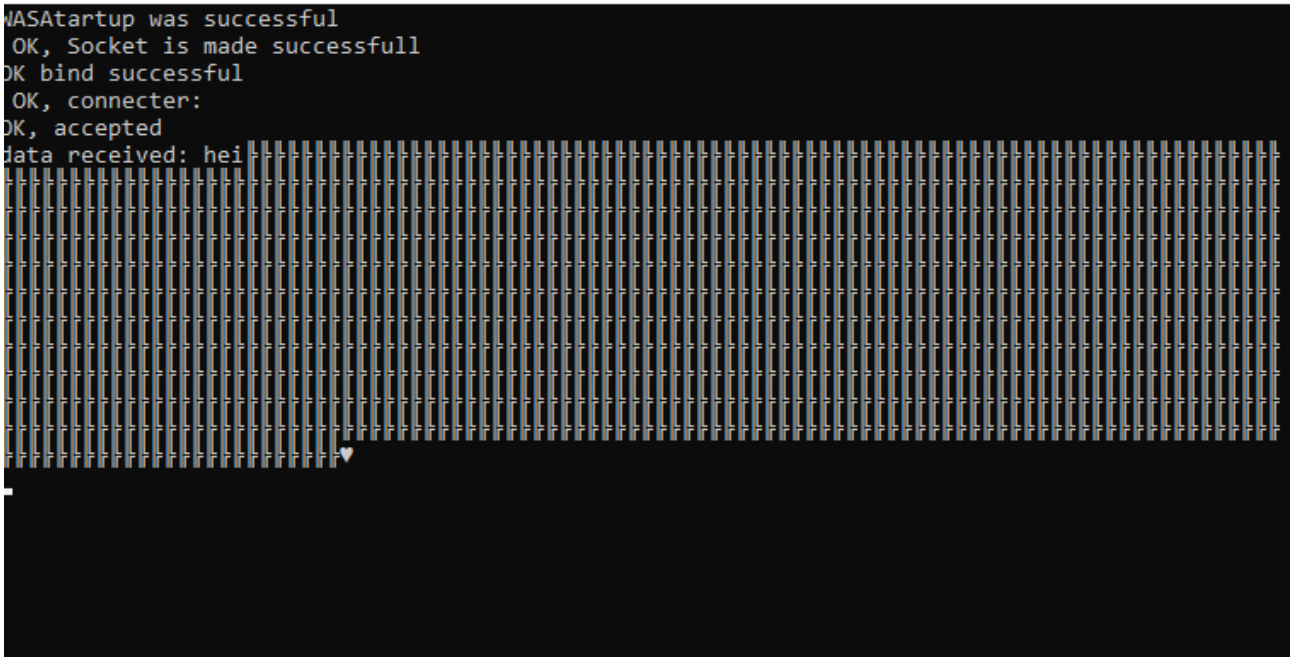
UT-8 test



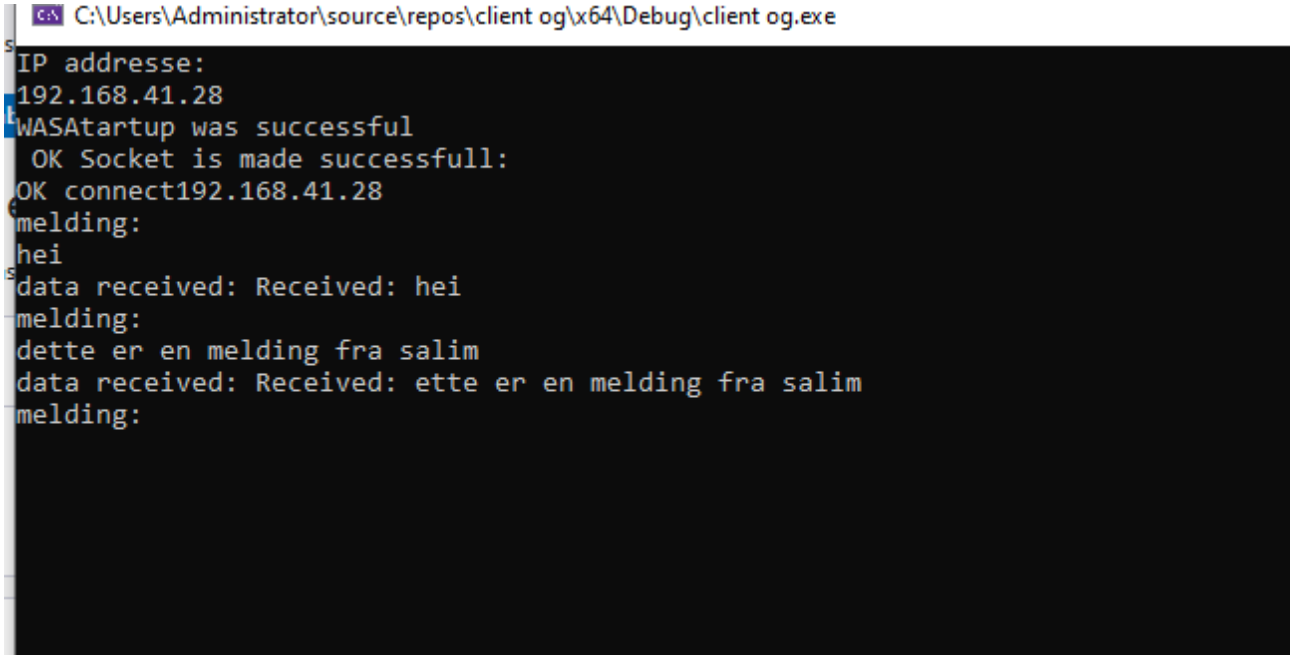
1024+ bytes, som man ser stopper den etter 1024 bytes.



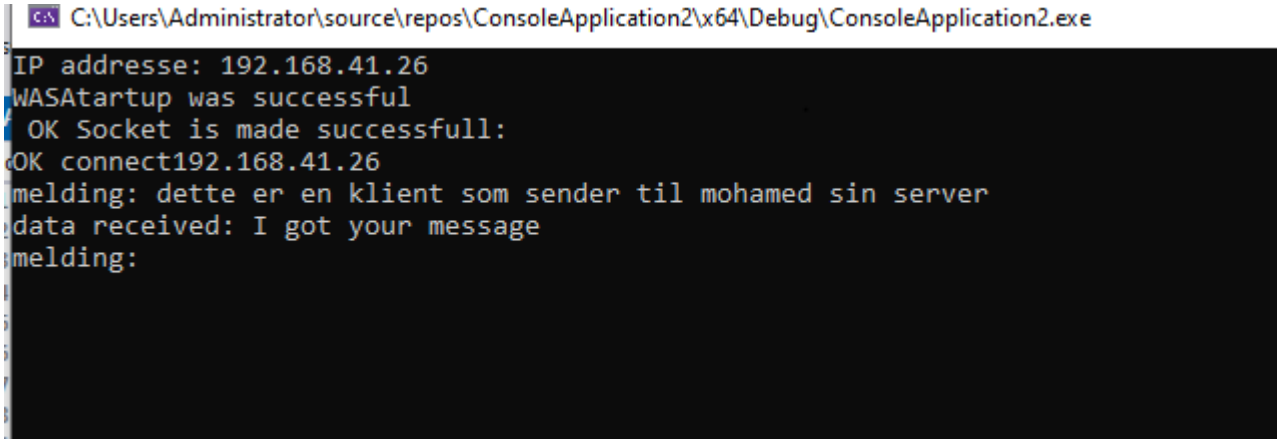
Uten buf[revSocket] = ’\0’. må ha den med for å kaste buffert «søppla».



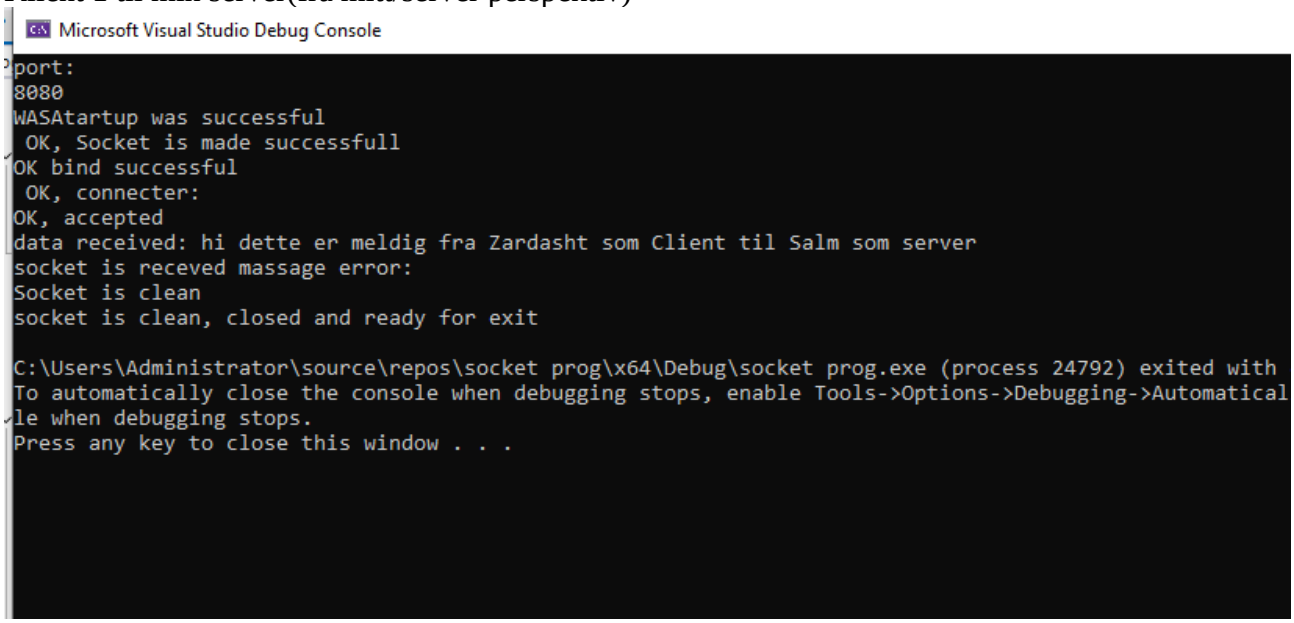
Min klient til en annen server 1



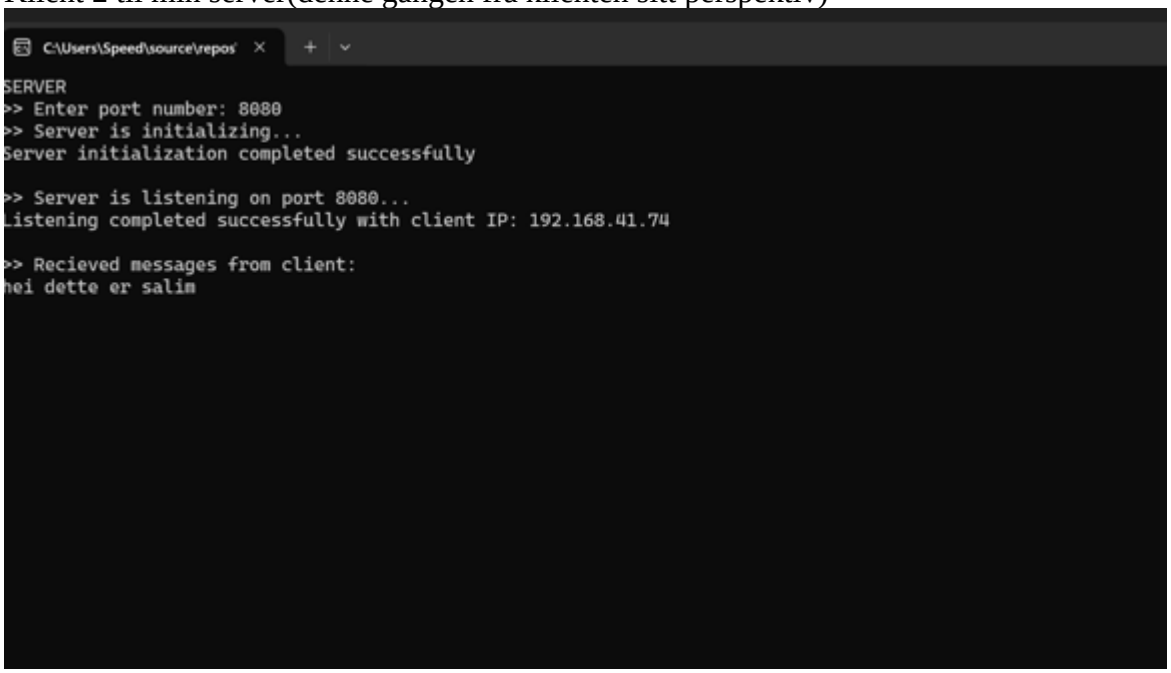
Min klient til en annen server 2



Klient 1 til min server(frå mitt/server perspektiv)



Klient 2 til min server(denne gangen fra klienten sitt perspektiv)



Sikkerhetsvurdering

Jeg skal gjøre noen vurderinger angående min fil. Et punkt meldingene som er sendt mellom dem ikke er beskyttet. Da kan må kryptere dataen. Jeg hadde endret send og recv for å ikke la andre ha tilgang til data som blir sendt mellom dem. Som regel er det relativt enkelt å gå inn i en annen sin server og legg inn hva som helst. Dette er en sikkerhetsrisiko. Man kan løse dette med en sjekk om å tillate dem inn. Jeg kunne ha løst dette med en sjekk på starten. Et annet sårbarhet jeg har er at jeg ikke vet hvem som kobler seg til serveren. Det kan være hvem som helst, så en forbedring kan være å bruke en ofstream for dette. Jeg kan også forbedre denne funksjonen ved å få logg til å se alle slags aktiviteter som skjer. Man kan også lage et slags «brannmur» som blokkerer uønsket aktivitet.

Konklusjon

jeg har gjort en svak jobb på signalhandler, siden jeg kun kaller dem til exit. Noe jeg har lært i etterkant. Jeg forstod hvor viktig det er å cleane alt opp og lukke socket der den skal. Ikke returnere 0 heller siden den stenger for tidlig. Jeg kunne også laget alt i ulike void, sånn at det hadde vært enklere og systematisk. Og kalle dem i main. Jeg hadde egentlig ikke bruk for cleansocket eller errorsocket, men hadde de for å gjøre det enklere.

Referanser

Her skal litteratur henvist til tidligere i rapporten listes opp, som vist i følgende eksempel:

[1] Day, N., «*C++ Network programming 1 part 1: Sockets*», Youtube, https://www.youtube.com/watch?v=gntyAFoZp-E
01.02.25.
[2] Ingar,P., *Tutorial. 10.02.25.ptptx* Presentasjon fra timen, 13.02.25.
[3] Chatgpt, http://www.kde.org/ (sist besøkt 20.02.2025)
[4] Notater fra timen.
.

- ## Vedlegg

Nedenfor har jeg lagt ut koden min for klient og server, med den sist nevnte først.

```
// DETTE ER MIN SERVER


#include <winsock2.h>
#include <ws2tcpip.h>
#include <fstream>
#include <csignal>
#include <iostream>
#pragma comment(lib,"Ws2_32.lib")

#include <csignal>
using namespace std;

//vise ulike hvordan sockaddr_in og in_addr er laget, selvom de er allerede i bibliotek

/*
typedef struct sockaddr_in {
    short sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
} SOCKADDR_IN, * PSOCKADDR_IN;

typedef struct in_addr {
    union {
        struct {
            u_char s_b1;
            u_char s_b2;
            u_char s_b3;
            u_char s_b4;
        } S_un_b;
        struct {
            u_short s_w1;
            u_short s_w2;
        } S_un_w;
        u_long S_addr;
```

```
            u_short s_w2;
        } S_un_w;
        u_long S_addr;
    } S_un;
} IN_ADDR,*PIN_ADDR,FAR*LPIN_ADDR;



//lage struct for sockaddr i bind og accept
typedef struct sockaddr {
    u_short sa_family;
    CHAR sa_data[14];
}SOCKADDR, * PSOCKADDR, * LPSOCKADDR;

*/

//log meldinger
void logg(const string& filename, const string& massage) {
    std::ofstream logName(filename, ios::app);
    if (logName.is_open()) {
        logName << massage << std::endl;
    }
    else {
        std::cout << "failed logg" << filename << std::endl;
    }
}

//signal hådtering SIGINT/SIGTERM
void signal_handler(int signal_number) {
    string fileName = "clientFile.txt";

    if (signal_number == SIGINT) {
        logg(fileName, "SIGINT (ctl + c)");
        std::cout << "sigint" << std::endl;
    }
    else if (signal_number == SIGTERM) {
        logg(fileName, "SIGINT (ctl + c)");
        std::cout << "sigterm" << std::endl;
```

```
        std::cout << "sigint" << std::endl;
    }
    else if (signal_number == SIGTERM) {
        logg(fileName, "SIGINT (ctl + c)");
        std::cout << "sigterm" << std::endl;
    }
    //program terminator
    exit(signal_number);
}

//lage kall for cleanup, feil og sist error, jeg vet det er upraktisk å lage en funksjon som allerede er laget, men det er letter for meg å kode det slikt
void cleanSocket() {
    string fileName = "clientFile.txt";
    logg(fileName, "Socket is cleaned up");
    WSACleanup();
    std::cout << "Socket is clean" << std::endl;
}

void errorSocket() {
    string fileName = "clientFile.txt";
    logg(fileName, "error is detected in socket");
    int errorSocket = WSAGetLastError();
    std::cout << "Socket error: " << errorSocket << std::endl;
}

int main() {

    //setter inn ip adressen som skal kobles til
    int port;
    std::cout << "port: " << std::endl;
    std::cin >> port;

    //setter navn på loggen
    string fileName = "clientFile.txt";
    logg(fileName, "Server witing for signal");

    //register signal SIGABRT/SIGINT handler
    signal(SIGABRT, signal_handler);
    signal(SIGINT, signal_handler);

    // starter opp WSA
```

```
    if (WSAStartup(MAKEWORD(2, 2), &wasData) != 0) {
        logg(fileName, "WSAStartup has not started up");
        std::cout << "WSAStartup failed" << std::endl;
        errorSocket();
        cleanSocket();
        return 1;
    }
    else {
        logg(fileName, "WASAtartup is running");
        std::cout << "WASAtartup was successful " << std::endl;
    }

    //lage socket
    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (serverSocket == INVALID_SOCKET) {
        std::cout << "socket error " << std::endl;
        errorSocket();
        cleanSocket();
        logg(fileName, "error in Serversocket ");
    }
    else {
        logg(fileName, "Serversocket is made");
        std::cout << " OK, Socket is made successfull" << std::endl;
    }

    //lage og binde socket
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080); //kan være 8080,8000,800,80
    auto serverAddr = inet_pton(AF_INET, "192.168.41.74", &server_addr.sin_addr); // "127.0.0.1" når jeg sender til meg selv

    //binde socket
    int bindSocket = bind(serverSocket, (SOCKADDR*)&server_addr, sizeof(server_addr));

    if (bindSocket == SOCKET_ERROR) {  //lage forbindelse med serversocket
        std::cout << "bind error " << std::endl;
        logg(fileName, "bind socket går ikke");
        errorSocket();
```

```
//binde socket
int bindSocket = bind(serverSocket, (SOCKADDR*)&server_addr, sizeof(server_addr));

if (bindSocket == SOCKET_ERROR) {  //lage forbindelse med serversocket
    std::cout << "bind error " << std::endl;
    logg(fileName, "bind socket går ikke");
    errorSocket();
    closesocket(serverSocket);
    cleanSocket();
    return 1;
}
else {
    logg(fileName, "socket did bind");
    std::cout << "OK bind successful " << std::endl;
}

//listen() funksjon
if (listen(serverSocket, 1) == SOCKET_ERROR) { //siden den skal kun være 1 kobling

    cleanSocket();
    return 1;
}
else {
    logg(fileName, "socket did bind");
    std::cout << "OK bind successful " << std::endl;
}

//listen() funksjon
if (listen(serverSocket, 1) == SOCKET_ERROR) { //siden den skal kun være 1 kobling
    std::cout << "did not connect: " << std::endl;
    logg(fileName, "server did not connect: ");
    errorSocket();
    closesocket(serverSocket);
    cleanSocket();
    return 1;
}
else {
    logg(fileName, "server is conncting" );
    std::cout << " OK, connecter: " << std::endl;
}

//lage og accept socket
struct sockaddr_in client_addr;
int addr_len = sizeof(client_addr);

SOCKET clientSocket = accept(serverSocket, (SOCKADDR*)&client_addr, &addr_len); //vet ikke hvem som kobler til
if (clientSocket == INVALID_SOCKET) {
    std::cout << "accept error:" << std::endl;
    logg(fileName, "server did not accept");
    errorSocket();
    closesocket(serverSocket);
    cleanSocket();
}
else {
    logg(fileName, "did accepted ");
    std::cout << "OK, accepted " << std::endl;
```

```
//while løkke for dialogen, recv og send
while (true) {
    //received
    char buf[1024];
    int recvSocket = recv(clientSocket, buf, 1024, 0);

    if (recvSocket <= 0) {
        std::cout << "socket is receved massage error: " << std::endl;
        logg(fileName, "received failed");
        break;
    }
    buf[recvSocket] = '\0'; //fjerner gammel data(søppel)
    std::cout << "data received: " << buf << std::endl;
    logg(fileName, "received successful");

    //melding svar fra server
    std::cout << "melding fra server: " << std::endl;
    char tekst[1024];
    std::cin.getline(tekst, 1024); //leser inn hver byte
    std::cin.ignore(); //fjerner uønsked data fra buffert

    // sende melding
    int sendSocket = send(clientSocket, tekst, strlen(tekst), 0);
    if (sendSocket == SOCKET_ERROR) {
        logg(fileName, "send socket failed");
        std::cout << "send error" << std::endl;
        errorSocket();
        break;
    }


//lukke alle sockets, end call
closesocket(serverSocket);
closesocket(clientSocket);
cleanSocket();
std::cout << "socket is clean, closed and ready for exit" << std::endl;
return 0;
```

```cpp
// DETTE ER MIN KLIENT


#include <winsock2.h>
#include <ws2tcpip.h>
#include <fstream>
#include <csignal>
#include <iostream>
#pragma comment(lib,"Ws2_32.lib")

#include <csignal>
using namespace std;

//log meldinger
void logg(const string& filename, const string& massage) {
    std::ofstream logName(filename, ios::app);
    if (logName.is_open()) {
        logName << massage << std::endl;
    }
    else {
        std::cout << "failed logg" << filename << std::endl;
    }
}

//signal hådtering SIGINT/SIGTERM
void signal_handler(int signal_number) {
    string fileName = "clientFile.txt";

    if (signal_number == SIGINT) {
        logg(fileName, "SIGINT (ctl + c)");
        std::cout << "sigint" << std::endl;
    }
    else if (signal_number == SIGTERM) {
        logg(fileName, "SIGINT (ctl + c)");
```

```
            logg(fileName, "SIGINT (ctl + c)");
            std::cout << "sigterm" << std::endl;
        }
        //program terminator
        exit(signal_number);
    }

    //lage kall for cleanup og error, igjen upraktisk
    void cleanSocket() {
        string fileName = "clientFile.txt";
        logg(fileName, "Socket is cleaned up");
        WSACleanup();
        std::cout << "Socket is clean" << std::endl;
    }

    void errorSocket() {
        string fileName = "clientFile.txt";
        logg(fileName, "error is detected in socket");
        int errorSocket = WSAGetLastError();
        std::cout << "Socket error: " << errorSocket << std::endl;
    }

    int main(){
        //setter inn ip adressen som skal kobles til
        string serverIP;
        std::cout << "IP addresse: " << std::endl;
        std::cin >> serverIP;

        //setter navn på loggen
        string fileName = "clientFile.txt";
        logg(fileName, "Server witing for signal");

        //register signal SIGABRT/SIGINT handler
        signal(SIGABRT, signal_handler);
```

```
    signal(SIGABRT, signal_handler);
    signal(SIGINT, signal_handler);

    // starter opp WSA
    WSADATA wasData;
    if (WSAStartup(MAKEWORD(2, 2), &wasData) != 0) {
        logg(fileName, "WSAStartup has not started up");
        std::cout << "WSAStartup failed. " << std::endl;
        errorSocket();
        cleanSocket();
        return 1;
    }
    else {
        logg(fileName, "WASAtartup is running");
        std::cout << "WASAtartup was successful " << std::endl;
    }

    //lage socket
    SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (clientSocket == INVALID_SOCKET) {
        std::cout << "socket error " << std::endl;
        errorSocket();
        cleanSocket();
        logg(fileName, "error in Clientsocket ");
    }
    else {
        logg(fileName, "Clientsocket is made");
        std::cout << " OK Socket is made successfull: " << std::endl;
    }

    //lage og koble til socket
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(8080); //kan være 8080,8000,800,80
```

```
server_addr.sin_port = htons(8080); //kan være 8080,8000,800,80
auto serverAddr = inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);
//server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

if (connect(clientSocket, (SOCKADDR*)&server_addr, sizeof(server_addr)) == SOCKET_ERROR) {  //lage forbindelse med serversocket
    std::cout << "connect error " << std::endl;
    logg(fileName, "connect socket går ikke");
    errorSocket();
    closesocket(clientSocket);
    cleanSocket();
    return 1;
}
else {
    logg(fileName, "socket is connected");
    std::cout << "OK connect" << serverIP <<std::endl;
}

// lage buffer til 1024 bytes, og teksten som skal skrives skal være 1024 bytes
char buf[1024];
char tekst[1024];

while (true) {

    //meldingen
    std::cout << "melding: " << std::endl;
    std::cin.ignore(); //fjerner uønsked data fra buffert
    std::cin.getline(tekst, 1024); //leser inn hver byte

    // sende melding
    int sendSocket = send(clientSocket, tekst, strlen(tekst), 0);
    if (sendSocket == SOCKET_ERROR) {
        logg(fileName, "send socket failed");
        std::cout << "send error" << std::endl;
        errorSocket();
```

```
    //mottatt
    int recvSocket = recv(clientSocket, buf, 1024, 0);

    //hvis du har mottatt
    if (recvSocket > 0) {
        buf[recvSocket] = '\0'; //kaster ut uønsket/resterende data, altså søpplet
        std::cout << "data received: " << buf << std::endl;
        logg(fileName, "data is recived ");
    }
    //hvis du ikke har mottat eller venter på
    if (recvSocket <= 0) {
        logg(fileName, "failed/waiting to receive");
        std::cout << "failed/waiting to receive: " << std::endl;
        errorSocket();
        break;
    }
}
//lukke alle sockets, end call
closesocket(clientSocket);
cleanSocket();
std::cout << "client is clean and closed" << std::endl;
logg(fileName, "socket is clean and closed");
return 0;
```