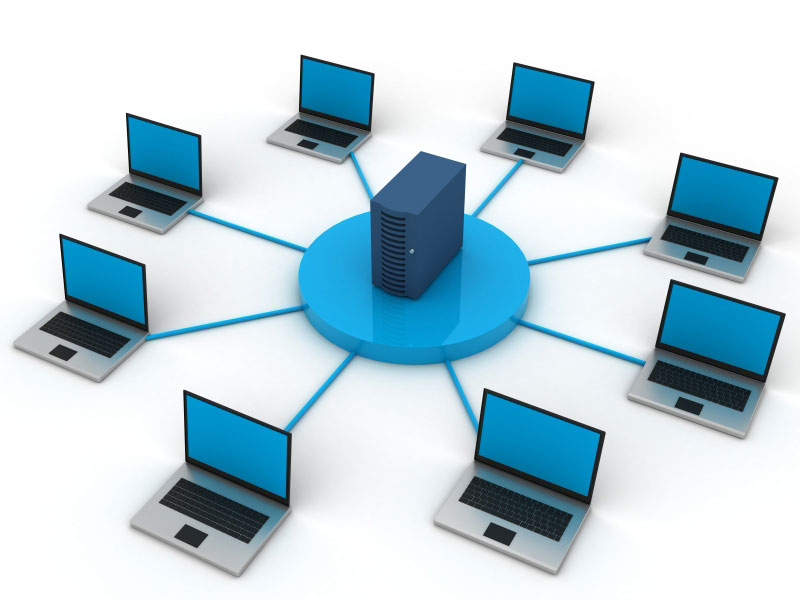
CHatProgramma

Door Simon de Graaf , Seppe Van Severen , Rob Willeart en Eli Jansegers



Inhoud

[1. Inleiding 2](#_Toc29402816)

[2. Communicatie over een Netwerk 3](#_Toc29402817)

[2.1 TCP 3](#_Toc29402818)

[2.2 Hoe gebruiken we TCP? 3](#_Toc29402819)

[2.3 Sockets 5](#_Toc29402820)

[3. Multithreading 6](#_Toc29402821)

[3.2 Wat zijn threads? 6](#_Toc29402822)

[3.3 Waarvoor gebruiken we threads? 6](#_Toc29402823)

[3.4 Hoe gebruiken we multithreading? 7](#_Toc29402824)

[4. Invoke 7](#_Toc29402825)

[4.1 Wat is invoke? 7](#_Toc29402826)

[5. Oefening 8](#_Toc29402827)

[5.1 Inleiding 8](#_Toc29402828)

[5.2 Klasse Users 8](#_Toc29402829)

# Inleiding

In het kader van de Geïntegreerde proef kozen wij de opdracht Netwerkgame. Dit houdt in een game met daarnaast een werkend chatprogramma die meerdere gebruikers tegelijkertijd met de game kunnen gebruiken om met elkaar te kunnen communiceren. Deze cursus zal meer duidelijkheid geven over het maken van dat chatprogramma.

Bij het maken van een chatprogramma heb je wat kennis nodig over bepaalde onderdelen die nodig zijn om dit te maken. Een chatprogramma bestaat uit twee onderdelen een client-gedeelte en een server-gedeelte. Van deze 2 is het de bedoeling dat jullie in onze oefening de client gaan maken. De server hebben wij immers al gemaakt voor jullie. Aan de hand van deze cursus gaan wij die onderdelen uitleggen zodat het maken van een chatprogramma makkelijker wordt.

In deze cursus wordt er eerst theorie uitgelegd aan de hand van voorbeelden en uitleg, daarnaast kunnen jullie tijdens de les die wij gaan geven ook nog vragen stellen. Op het einde van deze cursus staat er een oefening die jullie horen te maken met behulp van de theorie en de uitleg in de les.

Wij zullen o.a. uitleg geven over Sockets , TCP , UDP , **multithreading** etc. Deze termen zullen wij zodanig uitleggen dat u deze zal kunnen toepassen in praktijk.

# Communicatie over een Netwerk

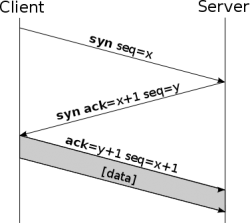
In dit hoofdstuk zullen we de middelen uitleggen om te kunnen communiceren over een netwerk. Dit zal essentieel zijn bij het maken van een client maar ook bij het maken van een server. Zonder dit onderdeel is het onmogelijk om data te kunnen versturen of ontvangen.

## 2.1 TCP

TCP of ***Transmission******Control******Protocol*** is een connectie-georiënteerd protocol dat veel gebruikt wordt voor het maken van een netwerkverbinding. Dit is echter slechts 1 manier waarop dit mogelijk is. De andere manier is UDP of ***User******Datagram******Protocol***. UDP is veel sneller dan TCP maar gaat de connectie tussen ***sender*** en ***reciever***niet controleren waardoor er data verloren kan gaan. Dit wordt gebruikt bij livestreams of ***First******Person******Shooter*****games**.

De primaire gedachten bij UDP is dus gegevens snel krijgen en bij TCP de garantie dat alle gegevens aankomen. Daarom gebruiken wij dus het principe TCP omdat dit ons de garantie geeft dat onze gegevens aankomen bij de ***server*** of ***client****.*

Omdat TCP zich op de transportlaag bevindt van het netwerkmodel moet er eerst een connectie gemaakt worden tussen een zender en ontvanger. TCP heeft hiervoor de ***Three******way******handshake***



De ***Three way handshake***zorgt voor een verbinding tussen een TCP-client en een TCP-server. De TCP-client stuurt een Pakket met een SYN-vlag (***synchronize***) naar de server. Als de server de verbinding accepteert, dan stuurt de server een pakket terug met zowel een SYN-vlag als een ACK-vlag (***acknowledge***). Als de client dit accepteert dan stuurt die vervolgens ook een pakket met een ACK-vlag naar de server. Bij het sluiten van de verbinding stuurt de server een pakket naar de client met een FIN-vlag (***final***) waarna de client een ACK-vlag terugstuurt en omgekeerd. Hierdoor zijn zowel de server als de **client** is nu op de hoogte van het sluiten van de verbinding.

## Hoe gebruiken we TCP?

TCP is de manier waarvan wij gebruikmaken bij het maken van onze client. Hier zijn een aantal belangrijke **datatypes** :

* **TcpClient** : Dit zorgt voor client connecties voor TCP netwerk diensten ( applicaties die draaien op de TCP-laag van de server)

Dit is hoe we een **TcpClient** zouden declareren met een bijhorend IP-adres en Poort(64553)





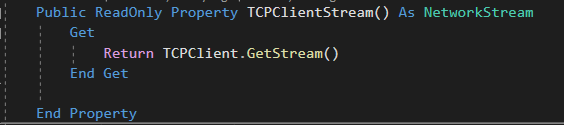
Hierbij horen ook een aantal **methodes** die we gebruiken bij het maken van de Client :

* Close() : zorgt ervoor dat de TCP verbindingen gesloten wordt , dit doe je meestal als een client zich wil verwijderen van het chatprogramma. Dan wordt de verbinding gesloten



* GetStream() : geeft de netwerkstroom terug die wordt gebruikt bij het versturen en ontvangen van data

in dit voorbeeld maken we een property aan die alleen gelezen kan worden , deze wordt gedeclareerd als **Networkstream** (het Datatype van een netwerkstroom) en krijgt vervolgens de Netwerkstroom van de TCP-client als waarde.



* **Connect**() **:** maakt een verbinding met een server of host door het IP-adres en de poort in te geven. Je doet hier hetzelfde als bij het declareren van je TCP-**client**.



* **StreamReader** : dit datatype leest karakters van een byte stream ( bv. Networkstream)

Het krijgt als parameter een datastroom waarvan hij de data kan lezen.

Als parameter wordt de netwerkstroom die we eerder hadden aangemaakt in **clientController** gebruikt.



Ook hierbij horen een aantal **methodes** die we gebruiken bij het maken van de Client :

* **Readline**() **:** leest de data van een stream en geeft deze data terug als string , hiermee kunnen we bv. Inkomende berichten onderscheppen en lezen.



Data is van het datatype string

* **StreamWriter :** dit datatype schrijft data weg op een stream. Het krijgt als parameter de stream waarna hij de data moet wegschrijven.



Ook hierbij horen een aantal **methodes** die we gebruiken bij het maken van de Client :

* **WriteLine**() : schrijft data weg naar de stream me daarachter een regel terminator(scheiding tussen verschillende regels) dit is om verschillende berichten te onderscheiden van elkaar.



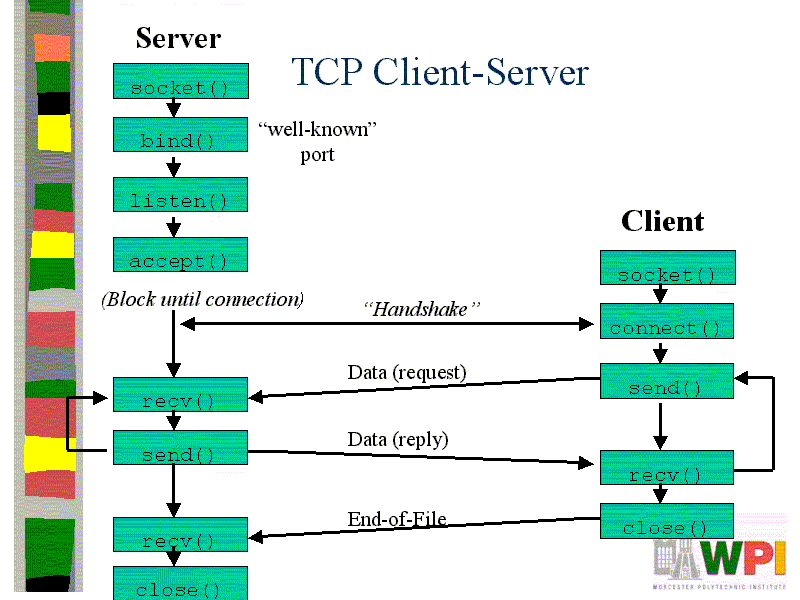
* **Flush**() : spoelt alle data door van de *Buffer* naar de datastream, de data kan dan gelezen worden door *streamreaders.*

Zonder deze methode zal de ontvanger de data nooit kunnen ontvangen.



## Sockets

In onze oefening gebruiken we TCP-clients om ervoor te zorgen dat we kunnen communiceren met andere computerprocessen. Maar er zijn ook andere mogelijkheden om dit te doen die wij niet gebruiken, namelijk **sockets**. Een internet **socket** is het Engelse woord voor een communicatie-eindpunt in een netwerk dat op het internetprotocol gebaseerd is. Elke socket wordt geïdentificeerd aan de hand van een combinatie van het IP-adres en de poortnummer, deze twee getallen worden gescheiden door een deelteken. Deze combinatie wordt ook wel socketadres genoemd. Om op het netwerk te kunnen communiceren maakt elk programma een socket aan. Aan de serverzijde worden hiervoor voor gedefinieerde poortnummers gebruikt. De **client** kiest zelf een willekeurig poortnummer (boven 1023) dat nog niet in gebruik is.



Dit schema geeft weer hoe een verbinding via sockets zou moeten verlopen.

Met de bijhorende methodes :

* **Socket**() : maakt een **socket**
* **Bind**() : koppelt een socket aan een IP-adres en poort nummer
* **Listen**() : Luistert naar inkomende Connecties
* **Accept**() : accepteert een connectie , deze methode stopt tot er een connectie is gemaakt
* **Connect**() : Connecteert de **socket** van de **Client** met de **server**
* **Send**() : verstuurt data over een verbinding
* **Recv**() : ontvangt data van een verbinding
* **Close**() : Sluit de verbinding

# Multithreading

* 1. Wat is Multithreading?

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijvingBij ***multithreaden*** wordt de taak verdeeld in kleinere delen, die kleinere delen noemen we ***threads***. Deze worden dan allemaal verstuurd en op het einde terug in hun originele vorm geplaatst. Het doel van ***multithreading*** is om meerdere taken tegelijk uit te kunnen voeren bv: het luisteren naar wat de ***cliënts*** sturen en ondertussen berichten kunnen ***displayen***.

In de afbeelding hierboven kan je zien dat één grote taak verdeeld is in 3 verschillende kleinere taken.

## Wat zijn threads?

We verdelen onze taken in ***threads***, maar wat zijn ***threads*** juist? ***Threads*** zijn kleine onderdelen van een grotere taak. Een voorbeeld ervan gaan we duidelijk zien in de oefening wanneer je de cliënt een bericht wilt sturen naar de server gebeurt dit met ***multithreading*** in de server. De server zal luisteren en ondertussen berichten versturen naar al de andere ***cliënts***. We gebruiken een thread om te kunnen luisteren en een thread om de berichten te kunnen versturen.

## Waarvoor gebruiken we threads?

Wij gaan ***threads*** gebruiken in de server voor het luisteren naar inkomende berichten van de ***cliënts*** en het doorsturen van de berichten van de cliënt naar alle andere verbonden ***cliënts***. Ook gebruiken we een ***thread*** op de ***cliënt***om te kunnen luisteren naar de server.

## Hoe gebruiken we multithreading?

Om te beginnen maak je een veld die je als *thread* declareert



Vervolgens maak je een nieuwe ***thread*** die ervoor zorgt dat een sub wordt uitgevoerd

(Hier is ‘***ConnectClient’*** de sub die wordt uitgevoerd)

Daarna zorg je ervoor dat de *thread* wordt gestart



In de server zit er niet veel tijd tussen het opstarten en het luisteren naar de *cliënt*, als er een bericht zou toekomen tussen die kleine periode dan gaat je code niet verder met het binnengekomen bericht, om dit te voorkomen gebruiken we ‘***Thread****.****Sleep*’**. Dit zorgt ervoor dat je code een aantal milliseconden wacht voor het verder gaat met de code.



# Invoke

## Wat is invoke?

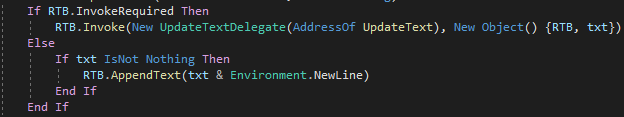
***Invoke*** is een functie die we gaan gebruiken wanneer we gebruik maken van meerdere ***threads***. ***Invoke*** kunnen we gebruiken als een thread niet de andere thread nodig heeft.

We gebruiken de functie **BeginInvoke**() in plaats van **invoke**() omdat ***invoke*** een ***deathlock*** kan veroorzaken. Ook moeten we deze gebruiken bij seriële poorten bij de **eventhandler** omdat de *thread* anders niet direct update.

Er zijn verschillende soorten functies bij ***invoke*** o.a.

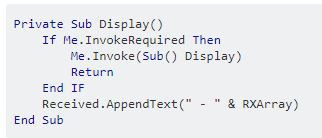
**Invoke**

**BeginIvoke**



Als het event wordt **geraised** gaat de RichTextBox

Je kan ook ***invoke***in een aparte sub zetten namelijk***Display***()



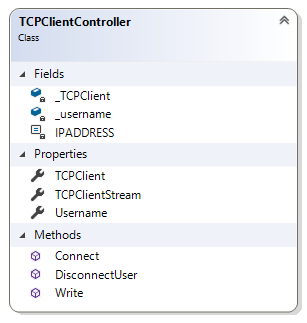
Dit kan je gebruiken voor het vergemakkelijken en het inkorten van de code.

# Oefening Chatprogram

Met deze oefening gaan we een chatprogramma maken dat kan gebruikt worden zodat *clients* met elkaar kunnen communiceren. Wat we voor deze oefening nodig hebben is een *client*, de server hebben wij al voor jullie gemaakt.

## 5.1 Client: solution

Maak een *Windows Forms Application* aan met de naam ***ChatClient.***

**TCPClientController**

* Maak 2 eigenschappen aan waarin de *TCPClient*, en *Username* kunnen bewaard worden. Declareer \_TCPClient als *TcpClient*. Maak ook een constante *IPaddress* en stel dit gelijk aan het IP-adres 127.0.0.1. Maak ook een *readonly* eigenschap TCPClientStream waarbij je de *netwerkstream* gaat teruggeven.

as String = "127.0.0.1"

as Tcpclient

as string

* In de sub *write* gaan we eerst aan validatie doen, dit is zodat we een bericht kunnen onderscheiden van een *username* op de *datastream*.

as string

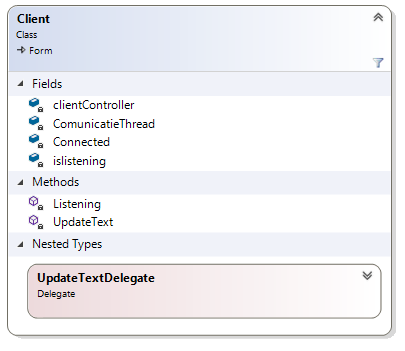
as NetworkStream

as Tcpclient

* In de sub *connect* ga je ervoor zorgen dat je kan connecteren, dit doen we aan de hand van de eigenschappen die we net hebben aangemaakt. Je gaat hier ook moeten gebruik maken van een *hostname* en een *port* deze *hostname* is *IPaddress* en de *hostname* is 64553.

(Message As String, Optional isUsername As Boolean = False)

as Boolean

* Maak een methode aan *DisconnectUser* die ervoor kan zorgen dat een user kan ontkoppelen van de chat.
* Zorg ervoor dat als je op de *send* knop drukt dat de tekst die je had getypt kan verstuurd worden.

As Thread

as Boolean

as Boolean

as New TCPClientController

* Als je op de *connect* knop drukt ga je eerst naar de username vragen en ga je ervoor zorgen dat je kan connecteren met de server. Probeer ook duidelijk te maken dat je bent geconnecteerd met de server aan de gebruiker.

(RTB As RichTextBox, txt As String)