ForkThreadsSockets

John meerten en Nicky van Riet  
T32

# Inhoudsopgave

Inhoud

[Inhoudsopgave 2](#_Toc500181992)

[Inleiding 3](#_Toc500181993)

[Process 4](#_Toc500181994)

[2.1 4](#_Toc500181995)

[2.2 5](#_Toc500181996)

[2.3 6](#_Toc500181997)

[2.4 7](#_Toc500181998)

[Conclusie 8](#_Toc500181999)

# Inleiding

In deze opdracht gaan we met tcp socket communicatie werken. En middelen om te kunnen multi-threaden hiervoor zullen we pthread en forks voor gebruiken. Door deze te combineren gaan we uiteindelijk een ‘chat service’ maken.

**Note:** De afbeeldingen in dit document zijn terug te vinden in de bijlages.

# Process

## 2.1

Eerst gaan we simpel beginnen. We gaan met een client data sturen naar de server en deze zet kleine letters om in hoofdletters en vice versa. Hierbij blijven de getallen onveranderd.

Om data te sturen maken we gebruik van de send() functie ( deze komt uit sys/sockets ). Hierbij moeten we meegeven met welke socket we aan het communiceren zijn. Deze is al voor ons aangemaakt. Daarna sturen we een string door, deze komt uit de parameters die je bij het opstarten van het programma meegeeft. Daarna geven we de lengte van de string aan. En als laats kan je eventueel een flag meegeven zoals ‘no wait’, hier maken we op het moment geen gebruik van.

Vervolgens om de data te ontvangen maken we gebruik van de recv() functie. Hierin geven we weer de socket waarvan we de data willen ontvangen. Een buffer waar we de data in gaan opslaan. De lengte van de buffer -1 (de lengte van de buffer is gedefineerd op 32), zodat we de string nog kunnen termineren. Dit doen we door het de ge ontvangen bytes van recv() op te vragen. En vervolgens op die plek een null terminator te schrijven. Als laatste kunnen we weer een flag toevoegen, maar hier maken we geen gebruik van.

Daarna moeten we code schrijven om de kleine letters naar hoofdletters te maken en vice versa. Dit kunnen we doen met de functie islower() deze geeft aan of het een kleine letter is. Als dit het geval is roepen we de functie touper() aan en deze maakt er vervolgens een hoofdletter van. Dit kunnen we omgedraaid ook doen met isupper() en tolower().

Nu kunnen we dus data versturen van de client naar de server. En de client krijgt dan vervolgens de geconverteerde tekst terug.

## 2.2

Nu gaan we een ‘echte’ chat client maken. We houden hierbij het principe aan dat de client eerst een bericht stuurt en dan pas de server en zo steeds om en om. Het bericht wordt uit de terminal gehaald. En om de chat te sluiten moet de server of de client het keyword ‘quit’ sturen.

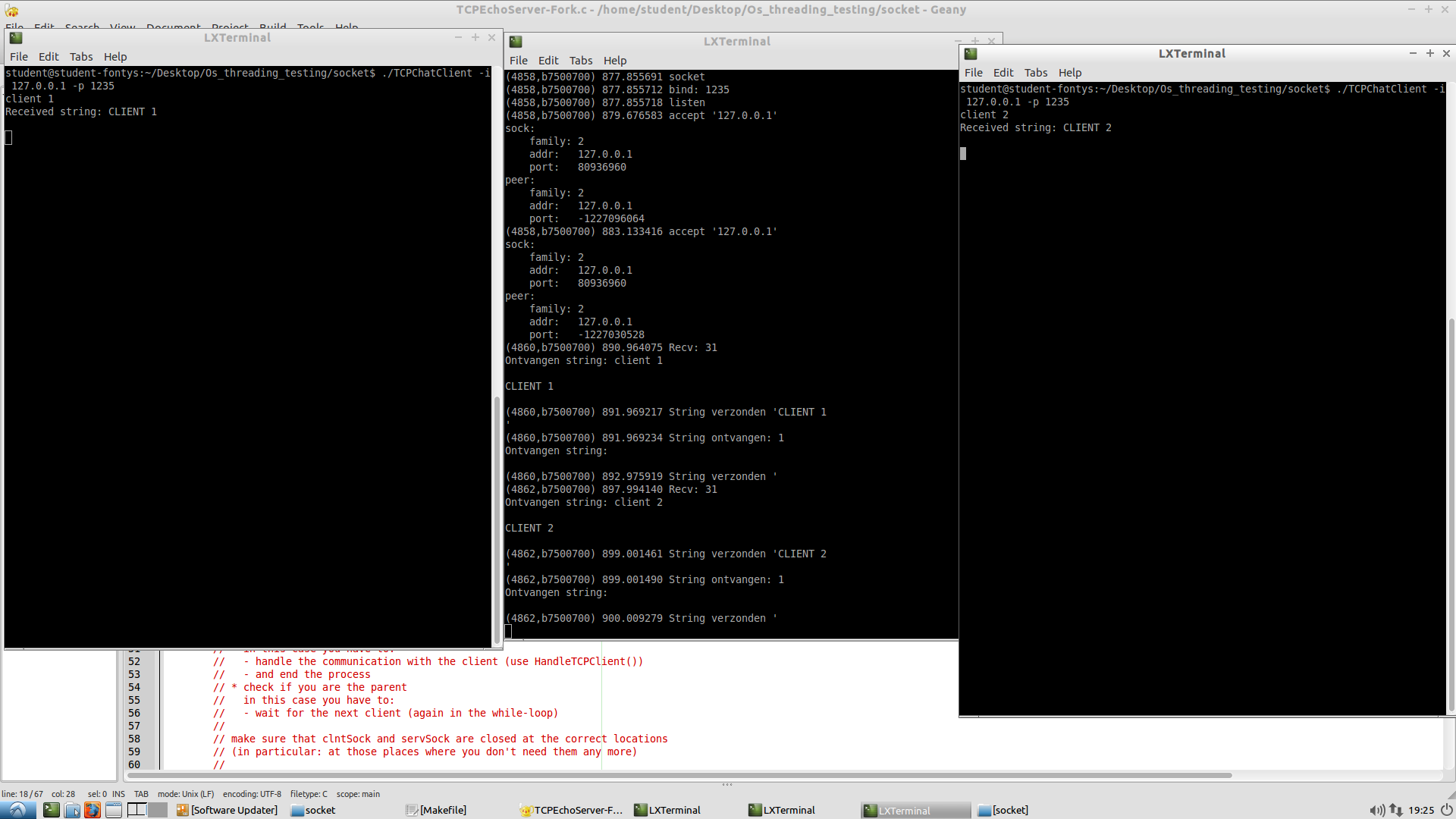
Het makkelijkste om dit te doen is door een switch te maken. Hierbij maken we een integer ‘mode’ aan. Deze start voor de server en client allebei in 0. Voor de server is mode 0 luisteren en voor de client is mode 0 schrijven. Zodra hun beurt is geweest wordt de mode met 1 verhoogd en gaat de server dus een bericht terug sturen en de client luisteren.

Om een bericht uit de terminal te halen gebruiken we de functie fgets(). Deze functie houd rekening met de buffersize die je mee geeft in tegenstelling tot de functie scanf(). Om vervolgens te kijken of ons keyword langs komt gebruiken we de functie strncmp(). De reden hiervoor is omdat de buffer altijd 31 bytes lang is, als we strcmp() zouden gebruiken zou ‘quit’ in de string kunnen staan maar omdat de rest van de bytes leeg is ziet strcmp() de string dan alsnog niet als gelijk met ‘quit’. Met strncmp() kunnen we aangeven dat we de eerste 4 bytes willen lezen waardoor we dus ‘quit’ wel goed uit kunnen lezen.

## 2.3

In dit gedeelte gaan we ervoor zorgen dat de server meerdere clients af kan handelen. Dit kunnen we makkelijk doen met de functie fork(). Als het process vraagt welk proces id hij heeft en hij is gesplits (een child) dan is zijn nummer 0. Als het ‘main’ process vraagt wat zijn nummer is is het altijd hoger dan 0. Hier kunnen we op inspelen met een simpele if statement.

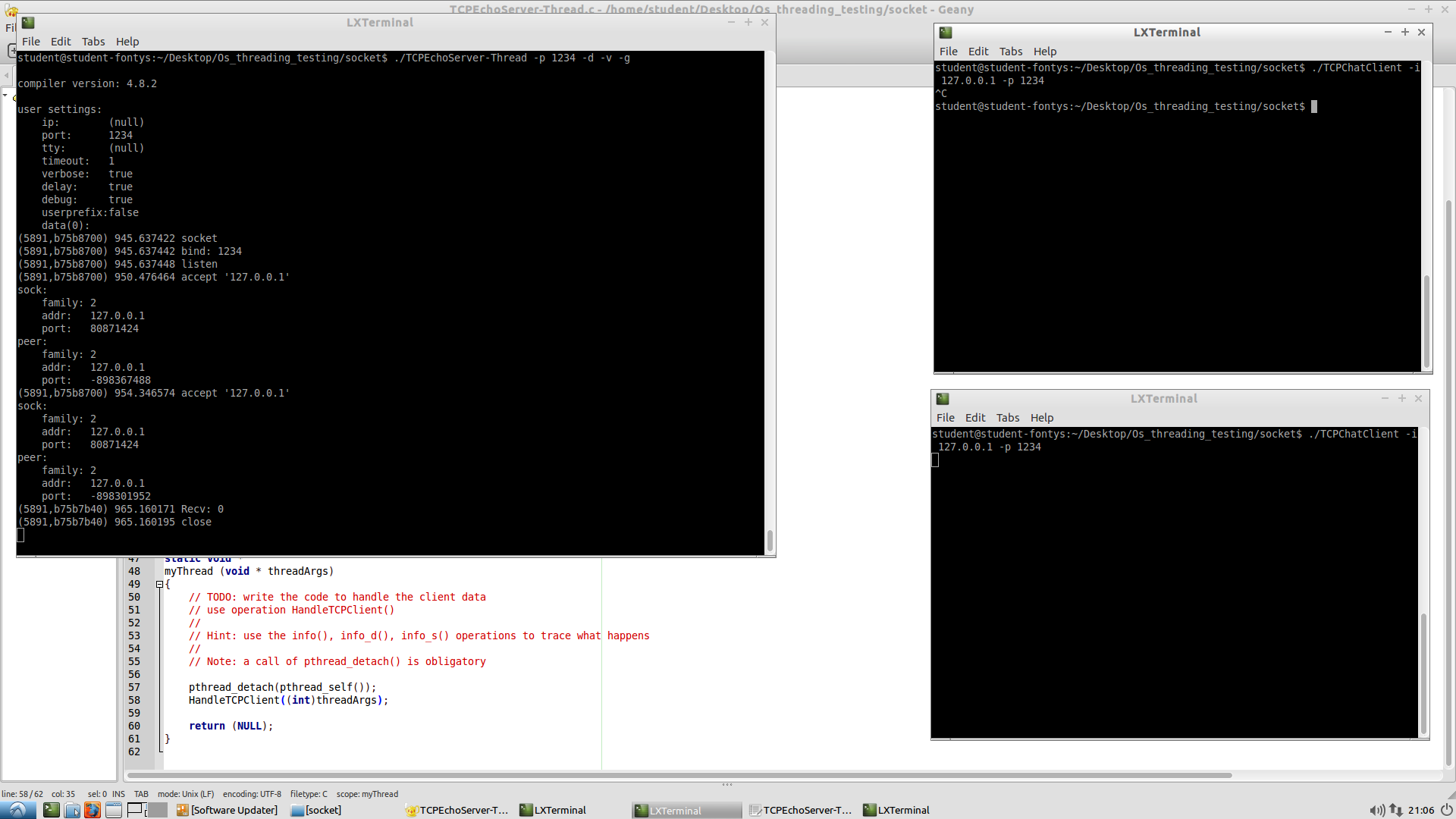
Wanneer het procces id 0 is en dus een child moet hij de tcp connectie afhandelen.(functie HandleTcpClient() ).   
Wanneer het procces id hoger is dan 0 en dus de main thread moet hij een nieuwe connectie opwachten.(functie AcceptTCPConnection() ).



## 2.4

Als laatste deel van de opdracht gaan we met threads werken ipv met forks. Dit kunnen we doen door de functie pthread\_create() te gebruiken.

Voor pthread\_create() heb je het volgende nodig. Een threadID, deze hoeft niet geinitialiseerd te zijn. Hiermee kan je later informatie op vragen of de thread weer laten joinen. Als tweede kan je een attribuut meegeven hiermee kan je bijvoorbeeld meegeven of een thread safe is of unsafe is, wij geven hier NULL mee zodat hij de default waardes gebruikt. Als 3de geef je een functie mee die de thread moet gaan uitvoeren. En als laatste geef je eventueel data mee in de vorm van een void pointer. In ons geval geven we hier de client socket in mee.



# Conclusie

We hebben een programma gemaakt waar we data van de ene terminal naar de andere terminal kunnen sturen. Hierbij hebben we gebruik gemaakt van tcp sockets. Daarna zijn we het programma stapgewijs uit gaan breiden door forks toe te passen en als laatste pthreads. Zodat de server meerdere client connecties kan onderhouden.