

# **Forking, Threads en Sockets**

Joran van Teeffelen & Bas Witters

## Inhoudsopgave

Inleiding.....	3
Sockets.....	3
Threads.....	3
Forks.....	3
Resultaten.....	4
Echo client/server.....	4
Chat client/server.....	6
Echo client/server fork.....	8
Echo client/server threads.....	9
Bronnen.....	9

Notitie: Alle code en screenshots zijn terug te vinden op onze Github pagina:  
<https://github.com/Pacmega/OS3/tree/master/Sockets%26Threads>

## Inleiding

In deze opdracht hebben we met forks, threads en sockets gewerkt. Voor de opdracht hebben we sockets gebruikt om communicatie tussen processen mogelijk te maken. We zijn begonnen met een chatclient en server op te zetten. Vervolgens hebben we de server aangepast om threads en forks te gebruiken en deze met elkaar te vergelijken.

## Sockets

Een internet socket is een eindpunt. Hiermee is het mogelijk om communicatie tussen 2 processen te creeren. Door een poortnummer en een IP-address kan een socket geïdentificeerd worden. Hierdoor kunnen meerdere processen, op een bepaalde computer(bijvoorbeeld), tegelijkertijd met elkaar praten.

## Threads

Een thread wordt gebruikt om taken binnen een programma te verdelen. Hierdoor kun je je processorgebruik binnen een programma optimaliseren om het programma zo snel mogelijk te laten werken.

## Forks

Bij een fork wordt er een kopie van een process gemaakt. Hierbij wordt zowel het programma als de inhoud gekopieerd. Denk hierbij aan variabelen en inhoud van geheugen die bij het programma horen. Vervolgens gaat zowel het nieuwe proces als het oude door met de code die na de fork aangeroepen word. Het verschil tussen de kopie en het origineel is het Process ID.

# Resultaten

## Echo client/server

Bij deze opdracht was het de bedoeling om met behulp van sockets een client tekst te laten sturen naar de server, waarbij de server de hoofd- en kleine letters verwisselde en vervolgens terugstuurde naar de client.

Gebruikte optie's om de client en server te draaien:

```
./TCPEchoServer -p 1234 -d -v -g  
./TCPEchoClient -i 192.168.3.103 -p 1234 -d -v -g aBcDeFgHiJkLmNoP 1234 XYz
```

*Opties voor de commands:*

-i staat voor ip. Hiermee geef je het IP address van de andere kant van de verbinding aan.

-p staat voor port. Hiermee geef je het gewenste poortnummer mee waar je over wil communiceren.

-d staat voor delay. Als je deze optie meegeeft in het programma, wordt er een sleep van 1 seconde aangeroepen. Dit kun je doen om ervoor te zorgen dat de communicatie tussen de server en de client trager verlopen om berichten makkelijker te kunnen lezen.

-v staat voor verbose. Hiermee print je de informatie over de socket na het verbinden. Dit geeft het volgende terug: Family: Dit geeft het gebruikte protocol terug; address: Het gebruikte address; port: De gebruikte poort.

-g debug-mode: Print de waarde van de opties van de gebruiker.

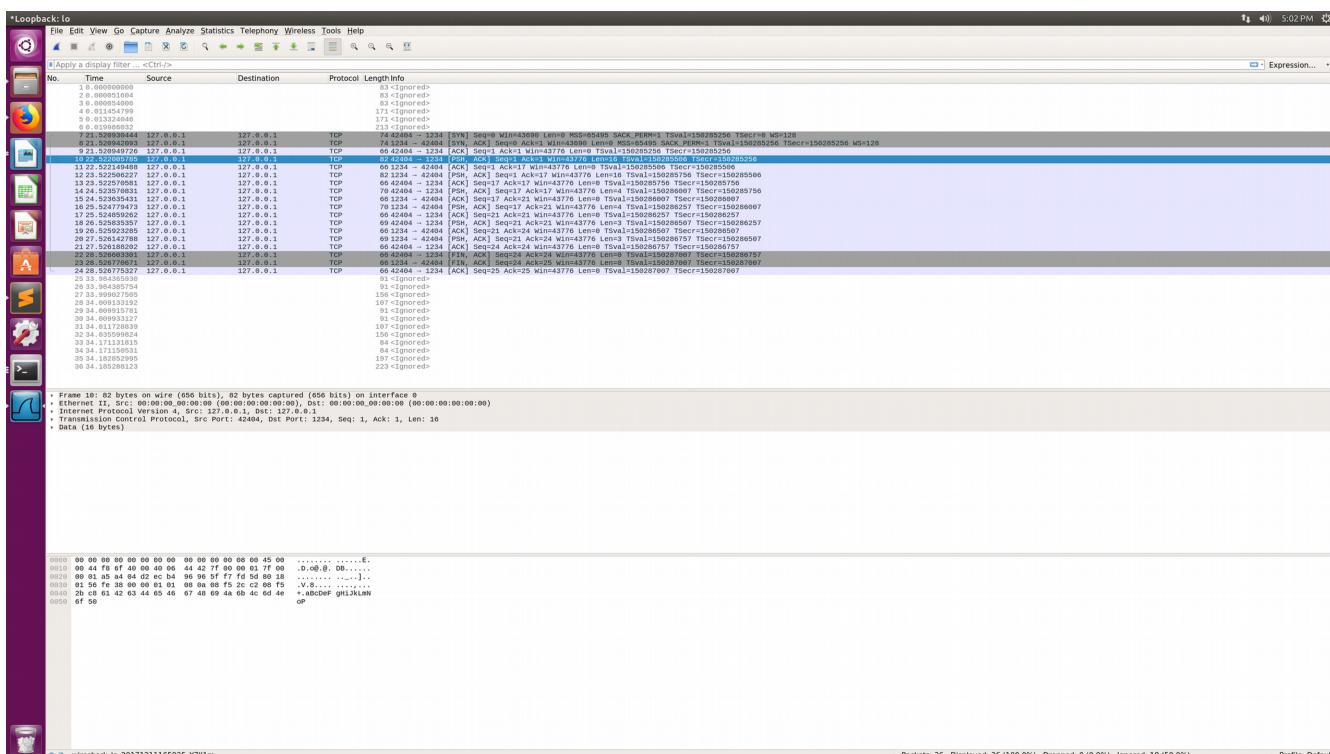
Deze optie's worden afgehandeld in auxilary. Er zijn nog meer optie's te vinden in het auxilary.c bestand, maar daar hebben wij geen gebruik van gemaakt.

```

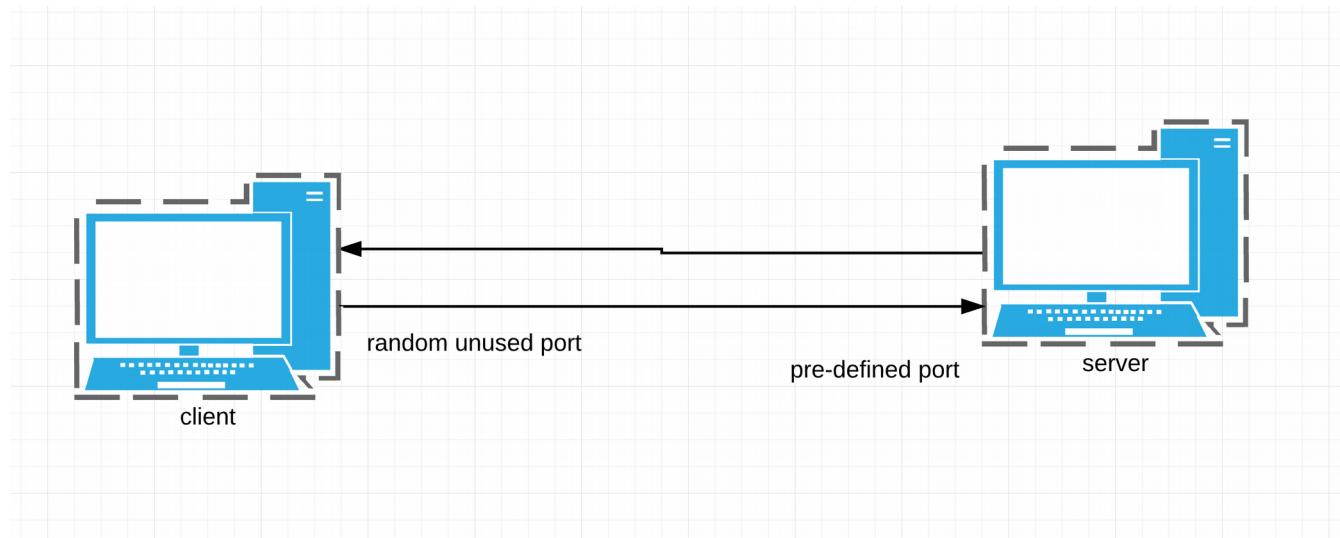
Terminal
succ@ubuntu:~/Documents/Fontys/OS3/Sockets&Threads/socket$ ./TCPEchoClient -l 127.0.0.1 -p 1234 -d -v -g abCdeFghijklmnop 1234 XYZ
compiler version: 5.4.0 20160609
user settings:
    ip:      127.0.0.1
    port:   1234
    tty:     (null)
    timeout: 1
    verbose: true
    delay:  true
    debug:   true
    userprefix:false
    data(3): 'abCdeFghijklmnop' '1234' 'XYZ'
    (12567,7f2143642700) 191.0151566 socket
    (12567,7f2143642700) 192.016185 connect
    sock:   Family: 2
            addr: 127.0.0.1
            port: 1528168448
    peer:   Family: 2
            addr: 127.0.0.1
            port: 86871424
peer:
    family: 2
    addr: 127.0.0.1
    port: 86871424
succ@ubuntu:~/Documents/Fontys/OS3/Sockets&Threads/socket$ ./TCPEchoServer -p 1234 -d -v -g
compiler version: 5.4.0 20160609
user settings:
    ip:      (null)
    port:   1234
    tty:     (null)
    timeout: 1
    verbose: true
    delay:  true
    debug:   true
    userprefix:false
    data(3): ''
    (12561,7f1ba2fe4700) 159.953736 socket
    (12561,7f1ba2fe4700) 159.953768 bind: 1234
    (12561,7f1ba2fe4700) 159.953771 listen
    (12561,7f1ba2fe4700) 192.016257 accept '127.0.0.1'
    sock:   family: 2
            addr: 127.0.0.1
            port: 1234
    peer:
        family: 2
        addr: 127.0.0.1
        port: 86871424
peer:
    family: 2
    addr: 127.0.0.1
    port: 86871424
succ@ubuntu:~/Documents/Fontys/OS3/Sockets&Threads/socket$ ./TCPEchoClient -l 127.0.0.1 -p 1234 -d -v -g abCdeFghijklmnop 1234 XYZ
    (12567,7f2143642700) 193.016711 verbose mode 'abCdeFghijklmnop'
    abCdeFghijklmnop
    (12567,7f2143642700) 195.018453 verbose mode '1234'
    (12567,7f2143642700) 197.020179 verbose mode 'XYZ'
    xyz
    (12567,7f2143642700) 199.021945 close & exit
succ@ubuntu:~/Documents/Fontys/OS3/Sockets&Threads/socket$ 

```

Het eerste wat opviel bij de opties die we kregen in de opdracht, is dat de server geen IP-address nodig heeft. Dit heeft te maken met dat de client de verbinding met de server aan gaat. De client moet weten waar hij naartoe moet gaan. De server stuurt berichten dan terug waar het vandaan komt. De server stuurt het bericht gewoon terug waar het vandaan komt. Ook hebben we tijdens het versturen/ontvangen wireshark gedraait om meer informatie te krijgen over de communicatie.

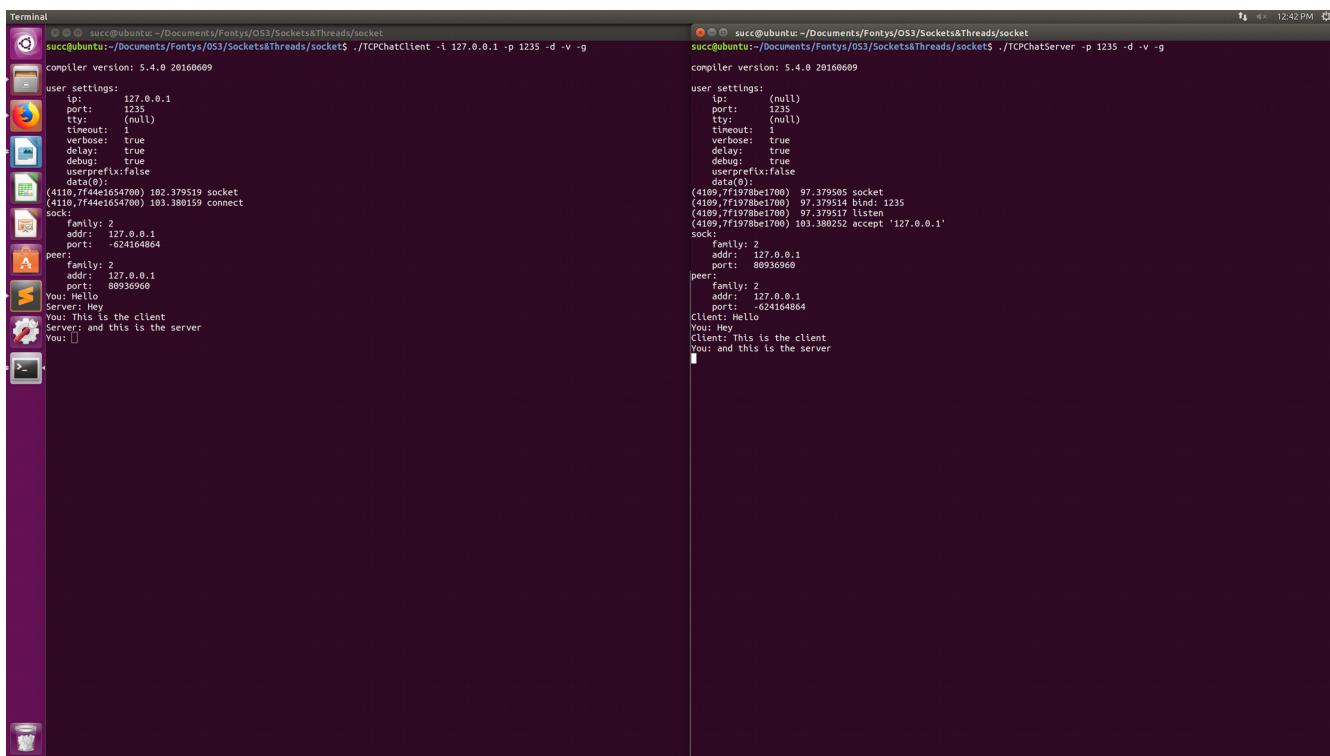


We waren verrast door de informatie, want er was nog een poort gebruikt naast poort 1234. Joran dacht eerst dat dit een willekeurige poort van de server was, omdat servers meerdere verbindingen tegelijkertijd aan moet kunnen gaan (al is dat hier niet het geval). Na wat onderzoek (wikipedia), zijn we erachter gekomen dat deze poort een willekeurige poort van de client was.



## Chat client/server

Voor deze opdracht was het de bedoeling om een chatapplicatie te maken. Hierbij verstuurde als eerste de client iets naar de server, waarop deze vervolgens iets terug kon sturen. Hiervoor hebben we 2 nieuwe bestanden gemaakt en de Makefile hierop aangepast. Deze Makefile bevat hiervoor 2 nieuwe commands (1 voor de client en 1 voor de server).



```
Terminal
● ● ● succ@ubuntu: ~/Documents/Fontys/OS3/Sockets&Threads/socket
succ@ubuntu:~/Documents/Fontys/OS3/Sockets&Threads$ ./TCPChatClient -i 127.0.0.1 -p 1235 -d -v -g
compiler version: 5.4.0 20160609
user settings:
  ip:      127.0.0.1
  port:   1235
  tty:    (null)
  timeout: 1
  verbose: true
  delay:  true
  debug:  true
  userprefix:false
  data(0):
  (4110,7f44e1654700) 102.379519 socket
  ((4110,7f44e1654700) 103.380159 connect
sock:
  family: 2
  addr: 127.0.0.1
  port: 624164864
peer:
  family: 2
  addr: 127.0.0.1
  port: 80936966
You: Hello
Server: Hey
You: This is the client
Server: and this is the server
You: [REDACTED]

● ● ● succ@ubuntu: ~/Documents/Fontys/OS3/Sockets&Threads/socket
succ@ubuntu:~/Documents/Fontys/OS3/Sockets&Threads$ ./TCPChatServer -p 1235 -d -v -g
compiler version: 5.4.0 20160609
user settings:
  ip:      (null)
  port:   1235
  tty:    (null)
  timeout: 1
  verbose: true
  delay:  true
  debug:  true
  userprefix:false
  data(0):
  (4109,7f1978be1700) 97.379505 socket
  ((4109,7f1978be1700) 97.379514 bind: 1235
  ((4109,7f1978be1700) 97.379517 listen
  ((4109,7f1978be1700) 103.380252 accept '127.0.0.1'
sock:
  family: 2
  addr: 127.0.0.1
  port: 80936966
peer:
  family: 2
  addr: 127.0.0.1
  port: 624164864
Client: Hello
You: Hey
Client: This is the client
You: and this is the server
```

Hierbij is ons niets nieuws opgevallen ten opzichte van de echo client en server.

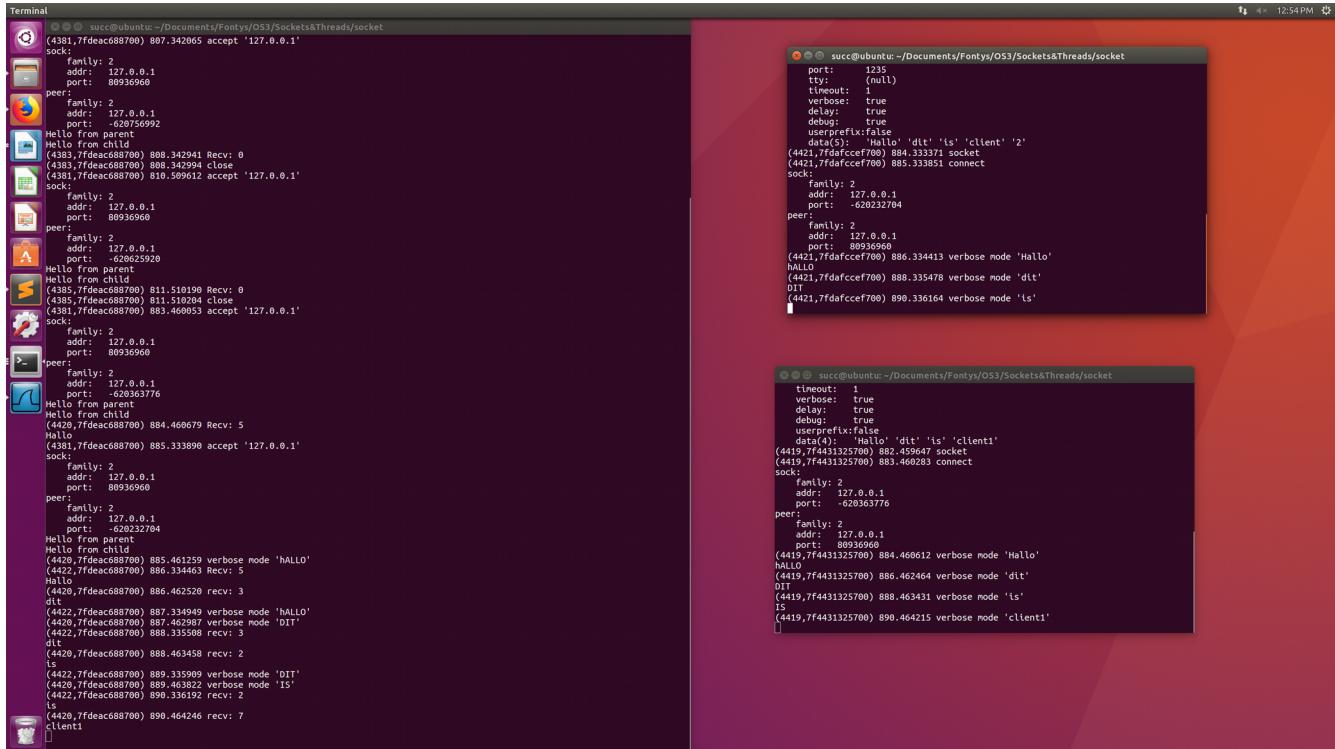
Gebruikte optie's:

./TCPChatClient -i 127.0.0.1 -p 1235 -d -v -g

./TCPChatServer -p 1235 -d -v -g

# Echo client/server fork

Met deze opdracht, hebben we bij de server voor elke verbinding met een client een apart process aangemaakt m.b.v. een fork. Hierdoor was het mogelijk om meerdere clients tegelijkertijd af te handelen.

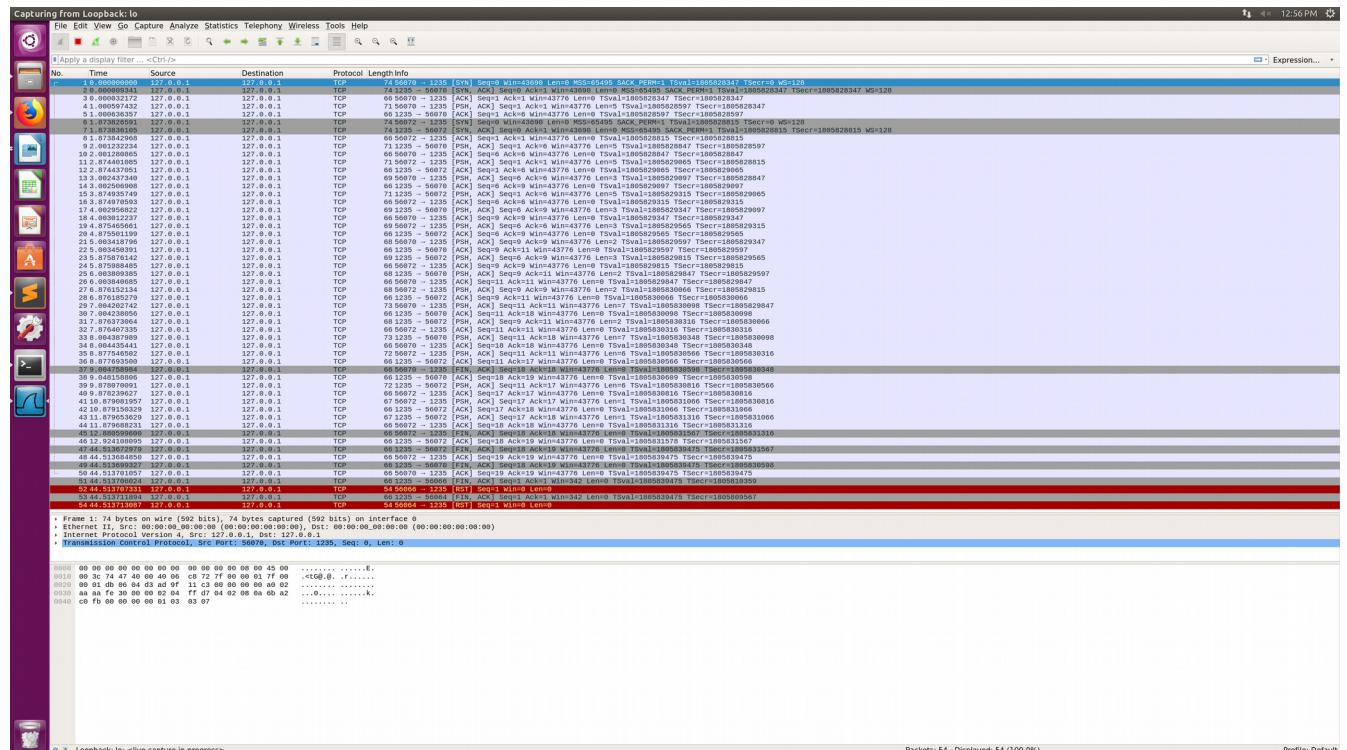


```
(4381,7fdeac688700) 887.342065 accept '127.0.0.1'
sock:
family: 2
addr: 127.0.0.1
port: 80936960
peer:
family: 2
addr: 127.0.0.1
port: 620232704
Hello from parent
Hello from child
(4383,7fdeac688700) 888.3c2941 Recv: 0
(4383,7fdeac688700) 888.342994 close
(4381,7fdeac688700) 810.509612 accept '127.0.0.1'
sock:
family: 2
addr: 127.0.0.1
port: 80936960
peer:
family: 2
addr: 127.0.0.1
port: -620232704
Hello from parent
Hello from child
(4385,7fdeac688700) 811.510109 Recv: 0
(4385,7fdeac688700) 811.510204 close
(4381,7fdeac688700) 883.460053 accept '127.0.0.1'
sock:
family: 2
addr: 127.0.0.1
port: 80936960
peer:
family: 2
addr: 127.0.0.1
port: -620232704
Hello from parent
Hello from child
(4420,7fdeac688700) 884.4d8679 Recv: 5
Hello
(4381,7fdeac688700) 885.333899 accept '127.0.0.1'
sock:
family: 2
addr: 127.0.0.1
port: 80936960
peer:
family: 2
addr: 127.0.0.1
port: -620232704
Hello from parent
Hello from child
(4422,7fdeac688700) 886.461359 verbose mode 'HALLO'
(4422,7fdeac688700) 886.334463 Recv: 5
Hello
(4420,7fdeac688700) 886.462520 recv: 3
dit
(4422,7fdeac688700) 887.334949 verbose mode 'halLO'
(4420,7fdeac688700) 887.462987 verbose mode 'DIT'
(4422,7fdeac688700) 889.335908 recv: 3
dit
(4420,7fdeac688700) 888.463458 recv: 2
is
(4422,7fdeac688700) 889.215939 verbose mode 'DIT'
(4420,7fdeac688700) 889.463822 verbose mode 'IS'
(4422,7fdeac688700) 890.336192 recv: 2
is
(4420,7fdeac688700) 890.464246 recv: 7
client1
```

```
port: 1235
tty: /null
timeout: 1
verbose: true
delay: false
debug: true
userprefix:false
data(5): 'Hello' <--> 'ls' <--> 'client' <--> '2'
(4421,7fdafcccf700) 884.333371 socket
(4421,7fdafcccf700) 885.333851 connect
sock:
family: 2
addr: 127.0.0.1
port: -620232704
peer:
family: 2
addr: 127.0.0.1
port: 80936960
Hello from parent
Hello from child
(4421,7fdafcccf700) 886.334413 verbose mode 'Hallo'
(4421,7fdafcccf700) 888.335478 verbose mode 'dit'
DIT
(4421,7fdafcccf700) 890.336164 verbose mode 'ls'
```

```
timeout: 1
verbose: true
delay: false
debug: true
userprefix:false
data(4): 'Hello' <--> 'dit' <--> 'is' <--> 'client1'
(4419,7f4431325700) 882.459647 socket
(4419,7f4431325700) 883.460283 connect
sock:
family: 2
addr: 127.0.0.1
port: -620232704
peer:
family: 2
addr: 127.0.0.1
port: 80936960
(4419,7f4431325700) 884.460612 verbose mode 'Hallo'
HALLO
(4419,7f4431325700) 886.462464 verbose mode 'dit'
DIT
(4419,7f4431325700) 888.463431 verbose mode 'is'
IS
(4419,7f4431325700) 890.464215 verbose mode 'client1'
```

Ook hebben we hierbij een wireshark ernaast laten lopen om te kijken of er nu weer een nieuwe poort bij was gekomen.



We zagen dat dit inderdaad het geval was. De server kreeg alle verzoeken netjes in poort 1235 en stuurde een antwoord terug naar de poort die bij de client hoorde.

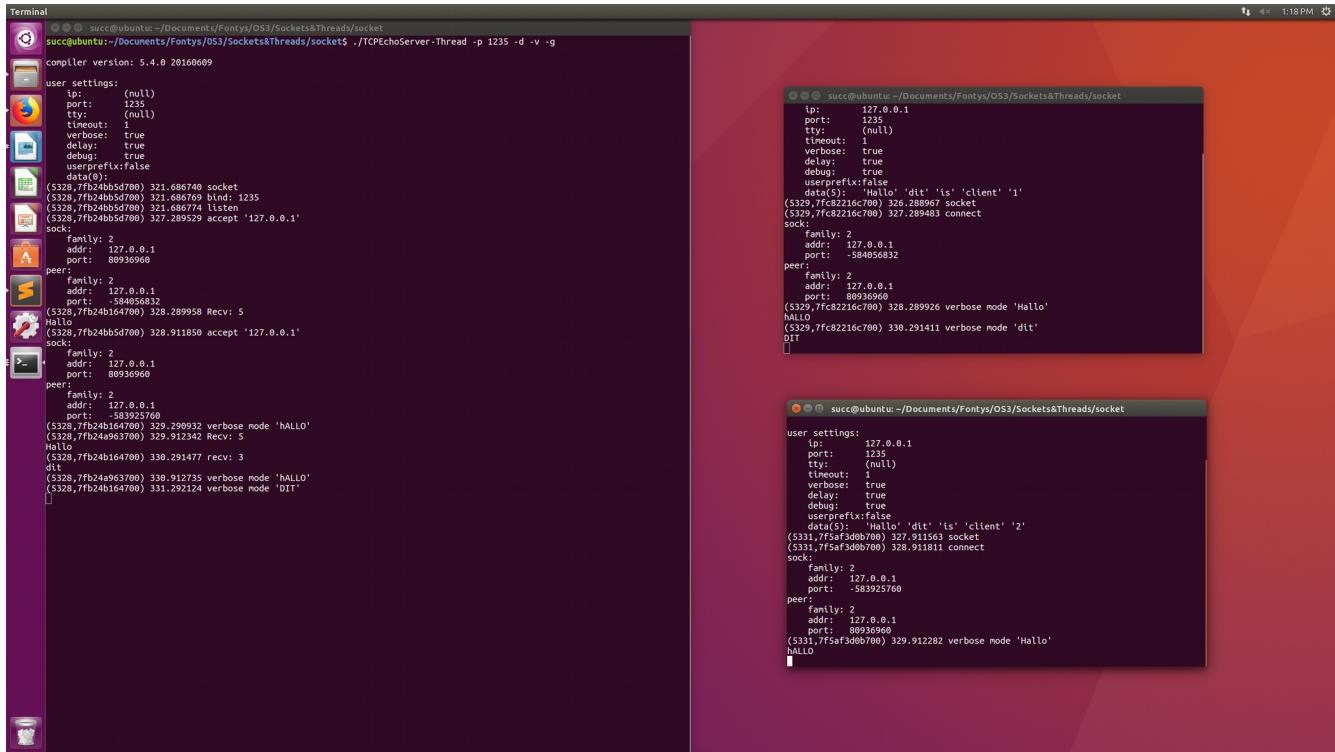
Gebruikte optie's:

./TCPEchoServer-Fork -p 1235 -d -v -g

./TCPEchoClient -i 127.0.0.1 -p 1235 -d -v -g Hallo dit is client[1/2]

# Echo client/server threads

Deze opdracht had dezelfde bedoeling als de vorige. Alleen maak je hier gebruik van threads in plaats van forks.



The image shows three terminal windows running on an Ubuntu desktop environment. Each window displays the output of a different process related to the TCP Echo client and server threads.

- Terminal 1:** Shows the execution of the TCP Echo Server. It lists user settings (ip: 127.0.0.1, port: 1235, tty: (null), timeout: 1, verbose: true, delay: true, debug: true, userprefix: false), socket details (family: 2, addr: 127.0.0.1, port: 8936960), and peer details (family: 2, addr: 127.0.0.1, port: 58405603). It also shows log entries for accepting connections and receiving data.
- Terminal 2:** Shows the execution of the TCP Echo Client. It lists user settings (ip: 127.0.0.1, port: 1235, tty: (null), timeout: 1, verbose: true, delay: true, debug: true, userprefix: false), socket details (family: 2, addr: 127.0.0.1, port: 8936960), and peer details (family: 2, addr: 127.0.0.1, port: 328.289958). It shows the client sending 'Hallo' and receiving data from the server.
- Terminal 3:** Shows the execution of another instance of the TCP Echo Client. It lists similar user settings and socket/peer details. It shows the client sending 'Hallo' and receiving data from the server.

Gebruikte optie's:

./TCPEchoServer-Thread -p 1235 -d -v -g

./TCPEchoClient -i 127.0.0.1 -p 1235 -d -v -g Hallo dit is client[1/2]

## Conclusie

Van deze serie opdrachten hebben we vooral geleerd hoe we, met behulp van sockets, processen met elkaar kunnen laten communiceren over het netwerk. Ook hebben we geleerd hoe we forks en threads kunnen gebruiken om programma's functioneler te maken. Voor de server die we hier gebruikt hebben lijkt het ons het beste om threads te gebruiken, omdat duplicatie van alle code en data in dit geval geen waarde toevoegd aan het programma. Daarom lijkt het ons handiger om aparte threads te gebruiken die de verbinding met de client afhandelen en vervolgens sluiten.

## Bronnen

Fork (informatica). (2014, Mei 14). Geraadpleegd op December 16, 2017, from  
[https://nl.wikipedia.org/wiki/Fork\\_\(informatica\)](https://nl.wikipedia.org/wiki/Fork_(informatica))

Stallings, W., W. (n.d.). Networking. In (pp. 1-25). Retrieved December 16, 2017, from  
<https://app.box.com/s/mbh0v0f6nx/file/242666102>.

Thread (informatica). (2017, November 24). Retrieved December 16, 2017, from  
[https://nl.wikipedia.org/wiki/Thread\\_\(informatica\)](https://nl.wikipedia.org/wiki/Thread_(informatica))

Internet socket. (2015, November 1). Geraadpleegd op December 14, 2017, from  
[https://nl.wikipedia.org/wiki/Internet\\_socket](https://nl.wikipedia.org/wiki/Internet_socket)