Onderzoeksverslag

– Client / Server –

Student:	A.W. Janisse
Studentnummer:	2213829
Instelling:	Fontys hogescholen te Eindhoven
Opleiding:	ICT & Technology
Docent:	Dhr. Erik Dortmans

Opdracht:	7, Client / Server
Datum:	12 april 2015

Inleiding

In deze opdracht gaat dieper in op Cliënt / Server communicatie in (fysiek) gedistribueerde systemen. De geijkte manier van communiceren is dan via het TCP/IP protocol. Om meerdere Clients tegelijk te kunnen bedienen kan de Server gebruik maken van forking of multi-threading.

Taken

In deze opdracht wordt de uitwerking van de volgende taken beschreven:

1. Echo cliënt/server. Compleet maken van aangeleverde code om een cliënt en server applicatie te maken. De cliënt stuurt data naar de server en de server stuurt de data weer terug. De server zal kleine letters omzetten in hoofdletters en grote letters omzetten in kleine letters.
2. Chat cliënt/server. Maken van twee programma's om te kunnen chatten tussen een client en een server.
3. Echo cliënt/server fork. Aanpassen van de chatserver zodat voor elke cliënt die zich aanmeldt de server een nieuw proces opstart die de communicatie met de cliënt afhandelt.
4. Echo cliënt/server threads. Aanpassen van de chatserver zodat voor elke cliënt die zich aanmeldt de server een nieuwe thread opstart die de communicatie met de cliënt afhandelt.
5. Netwerk analyse. Met het programma *wireshak* netwerk data analyseren.

Uitwerking taak 1

Betekenis parameters

De server en de cliënt kunnen worden opgestart met verschillende parameters. Als -h als parameter wordt gegeven dan wordt er weergegeven welke parameters er gebruikt kunnen worden. Dit ziet er als volgt uit:

```
options:
    -i <ip>
    -y <tty-name>
    -t <timeout>
    -p <port>
    -f <fork-max>
    -d          delay operation
    -g          debug info
    -u          user prefix
    -v          verbose
    <data>*
```

De gegeven parameters hebben de volgende betekenis:

-i	Deze parameter wordt door de cliënt gebruikt om het IP adres van de server op te geven. Bijvoorbeeld: -i 127.0.0.1
-y	Met deze parameter kan een bestandsnaam worden meegegeven. Als dit wordt gedaan zullen verschillende interne functies hun gegevens in het bestand schijven in plaats van naar stdout.
-t	Ongebruikt.
-p	Wordt gebruikt om het poortnummer op te geven. Client en Server gebruiken deze poort bij het opzetten van de verbinding. Bijvoorbeeld: -p 1234
-f	Ongebruikt.
-d	Als deze parameter wordt gebruikt zal op verschillende plaatsen in de programma's een vertragingstijd van 1 seconde worden gebruikt.
-g	Als deze optie wordt gebruikt wordt extra informatie weergegeven. Zie afbeelding 1.
-u	Als deze optie wordt gebruikt zullen diverse interne functie hun informatie weergeven met extra tijdsinformatie. Deze tijdsinformatie heeft de volgende vorm: hh:mm:ss.
-v	Als deze optie wordt gebruikt dan zal op verschillende plaatsen in beide programma's extra informatie worden weergegeven.
<data>*	De parameter wordt door de cliënt gebruikt om een of meerdere strings op te geven d ie afzonderlijk naar de server zullen worden gestuurd.

```
compiler version: 4.8.2

user settings:
    ip:         127.0.0.1
    port:       1234
    tty:        (null)
    timeout:    1
    verbose:    true
    delay:      false
    debug:      true
    userprefix:false
    data(3):    'aBcDeFgHiJkLmNoP' '1234' 'XYz'
```

Afbeelding 1

Uitbreidingen server

Voor de server zijn de onderstaande aanpassingen.

in de file 'Auxiliary.h' zijn de volgende functie definities toegevoegd:

```
29  extern void add_nt(char * s, int len);
30  extern void invert_case(char * s);
```

En deze zijn in de file 'Auxiliary.c' als volgt geïmplementeerd:

```
58  void
59  add_nt(char * s, int len)
60  {
61      // add 0 terminator to a string
62      s [len] = '\0';
63  }
```

```
68  void
69  invert_case(char * s)
70  {
71     char ch;
72     uint i;
73
74     for (i = 0; i < strlen(s); ++i)
75     {
76          ch = s[i];
77          if (ch >= 'A' && ch <= 'Z')
78          {
79              s[i] = s[i] + 32;
80          }
81          else if (ch >= 'a' && ch <= 'z')
82          {
83              s[i] = s[i] - 32;
84          }
85     }
86  }
```

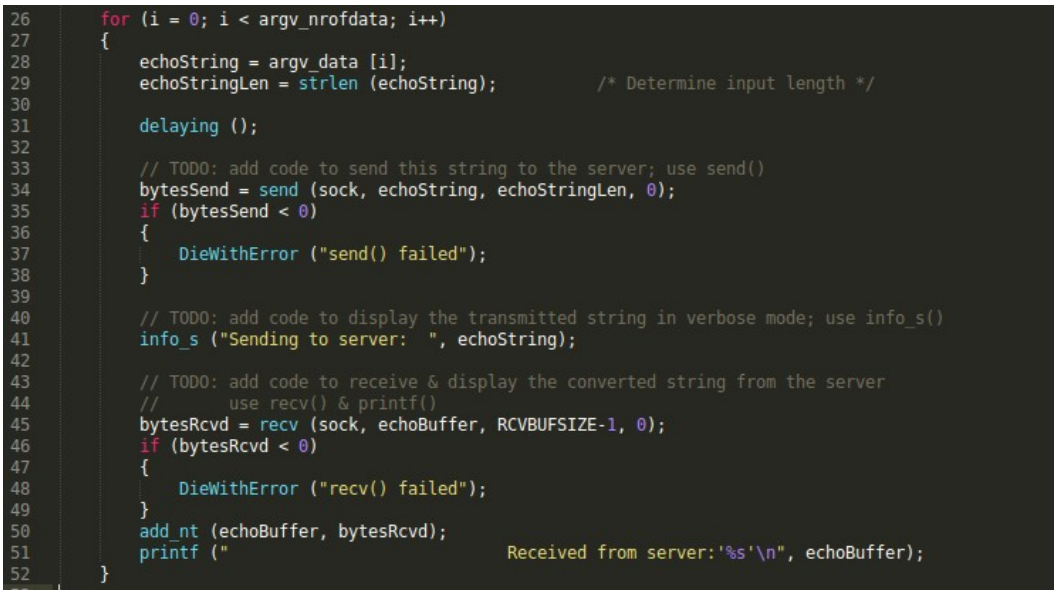
In de file 'HandleTCPClient.c' zijn de volgende regels toegevoegd:

```
29          // TODO: add code to print the received string; use printf()
30          add_nt(echoBuffer, recvMsgSize);
31          printf("Received from client: %s\n", echoBuffer);
32
33          // TODO: add code to convert the upper/lower chars of the received string
34          invert_case(echoBuffer);
```

```
43
44          // TODO: add code to display the transmitted string in verbose mode; use info_s()
45          info_s("Inverted from client: ", echoBuffer);
46
```

Uitbreidingen client

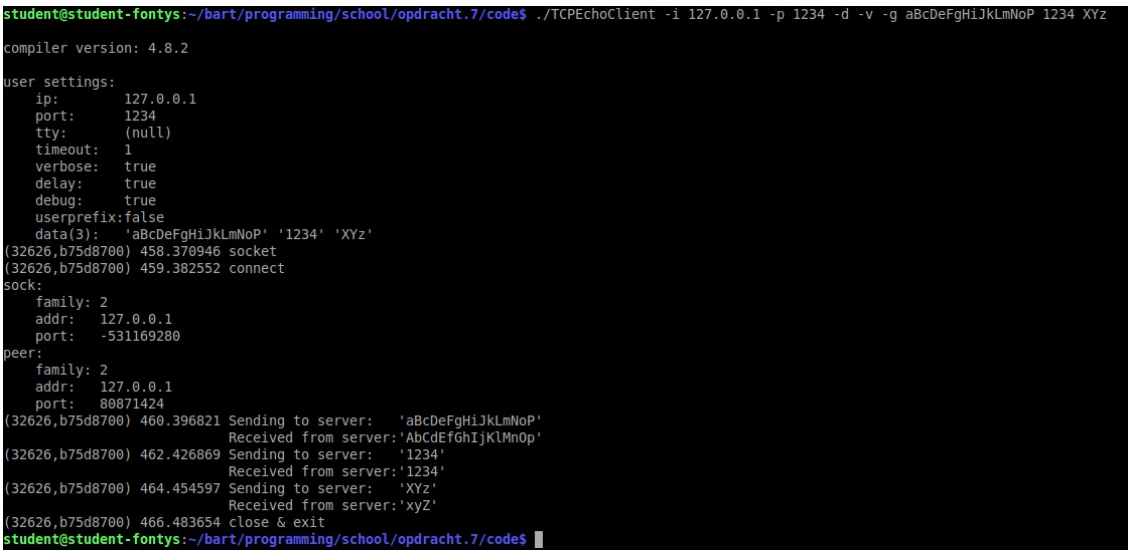
In de file 'TCPEchoClient.c' is de functie main als volgt aangepast:



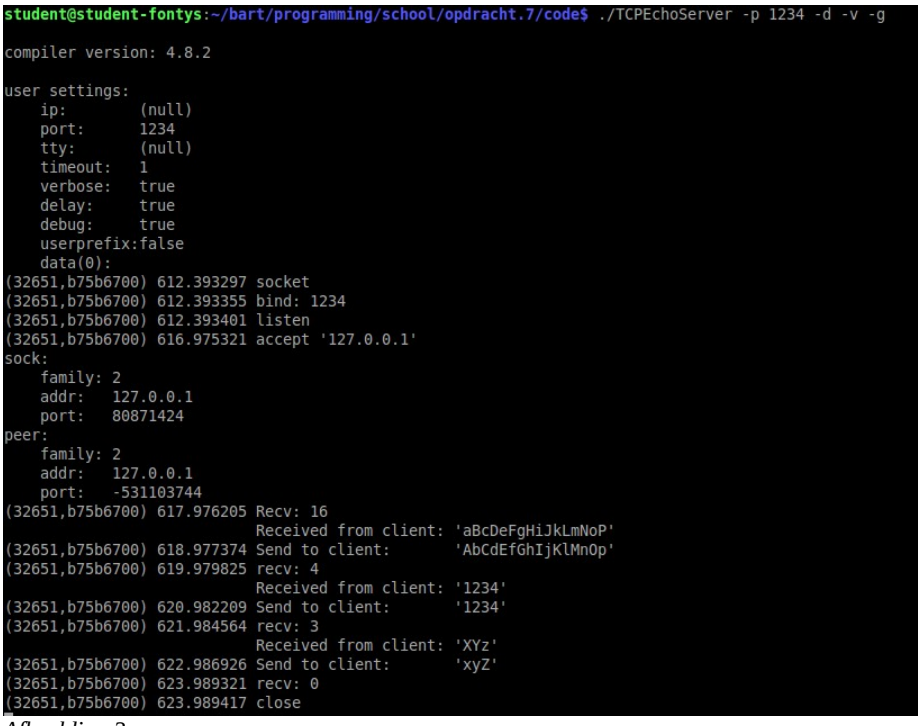
```
26      for (i = 0; i < argv_nrofdata; i++)
27      {
28          echoString = argv_data [i];
29          echoStringLen = strlen (echoString);          /* Determine input length */
30
31          delaying ();
32
33          // TODO: add code to send this string to the server; use send()
34          bytesSend = send (sock, echoString, echoStringLen, 0);
35          if (bytesSend < 0)
36          {
37              DieWithError ("send() failed");
38          }
39
40          // TODO: add code to display the transmitted string in verbose mode; use info_s()
41          info_s ("Sending to server:  ", echoString);
42
43          // TODO: add code to receive & display the converted string from the server
44          //       use recv() & printf()
45          bytesRcvd = recv (sock, echoBuffer, RCVBUFSIZE-1, 0);
46          if (bytesRcvd < 0)
47          {
48              DieWithError ("recv() failed");
49          }
50          add_nt (echoBuffer, bytesRcvd);
51          printf ("                          Received from server:'%s'\n", echoBuffer);
52      }
```

Resultaten

Afbeelding 2 laat het resultaat van uitvoeren van de client zien. Afbeelding 3 laat het resultaat van uitvoeren van de server zien.



Afbeelding 2



Afbeelding 3

Uitwerking taak 2

Voor deze taak is het de bedoeling om een nieuwe client en een nieuwe server te maken. De client en de server lezen beide een string van de terminal en sturen deze over. Om de client en de server af te sluiten kan het commando ' Quit' worden gegeven.

Onderstaande code is de ChatClientt.

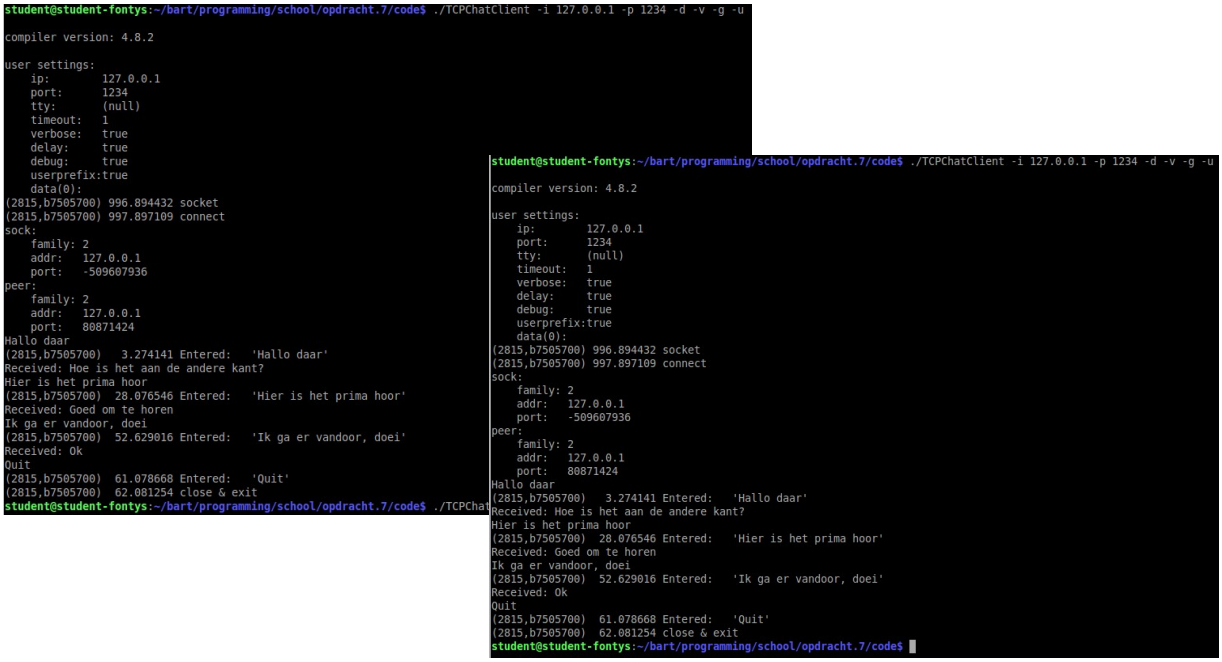
```
21
22      sock = CreateTCPClientSocket (argv_ip, argv_port);
23
24      while (strcmp (line, "Quit") != 0)
25      {
26          /* Send message to server */
27          fgets (line, sizeof (line), stdin);
28          remove_nl (line);
29          info_s ("Entered:  ", line);
30
31          bytesSend = send (sock, line, strlen (line), 0);
32          if (bytesSend < 0)
33          {
34              DieWithError ("send() failed");
35          }
36
37          /* Receive message from server */
38          if(strcmp (line, "Quit") != 0)
39          {
40              bytesRcvd = recv (sock, echoBuffer, RCVBUFSIZE-1, 0);
41              if (bytesRcvd < 0)
42              {
43                  DieWithError ("recv() failed");
44              }
45              if (bytesRcvd > 0)
46              {
47                  add_nt (echoBuffer, bytesRcvd);
48                  printf ("Received: %s\n", echoBuffer);
49              }
50          }
51      }
52
```

Onderstaande code is onderdeel van de ChatServer. In de server wordt het werk gedaan in de subroutine HandleTCPChatClient.

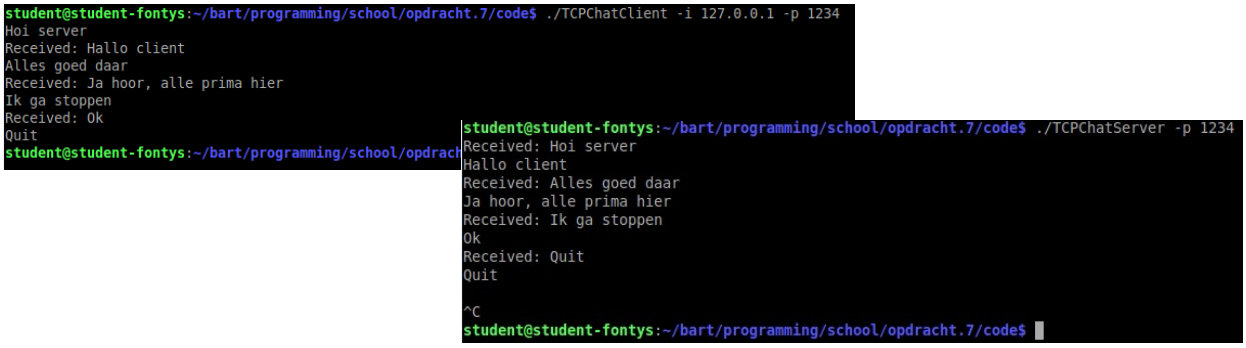
```
12  void HandleTCPChatClient (int clntSocket)
13  {
14      // 'clntSocket' is obtained from AcceptTCPConnection()
15
16      char    echoBuffer[RCVBUFSIZE];     /* Buffer for echo string */
17      int     bytesRcvd   = 0;            /* Size of received message */
18      int     bytesSend   = 0;            /* Bytes read in single send() */
19      char    line[80];                   /* String entered by user to send to server */
20
21      while (strcmp (line, "Quit") != 0)
22      {
23
24          while (bytesRcvd == 0)
25          {
26              /* Receive message from client */
27              bytesRcvd = recv (clntSocket, echoBuffer, RCVBUFSIZE-1, 0);
28              if (bytesRcvd < 0)
29              {
30                  DieWithError ("recv() failed");
31              }
32              if (bytesRcvd > 0)
33              {
34                  info_d ("Recv", bytesRcvd);
35                  add_nt (echoBuffer, bytesRcvd);
36                  printf ("Received: %s\n", echoBuffer);
37              }
38          }
39
40          /* Send message to client */
41          fgets (line, sizeof (line), stdin);
42          remove_nl (line);
43          info_s ("Entered:  ", line);
44
45          bytesSend = send (clntSocket, line, strlen (line), 0);
46          if (bytesSend < 0)
47          {
48              DieWithError ("send() failed");
49          }
50
51          bytesRcvd = 0;
52      }
```

Resultaten

Met verschillende parameters



Zonder parameters



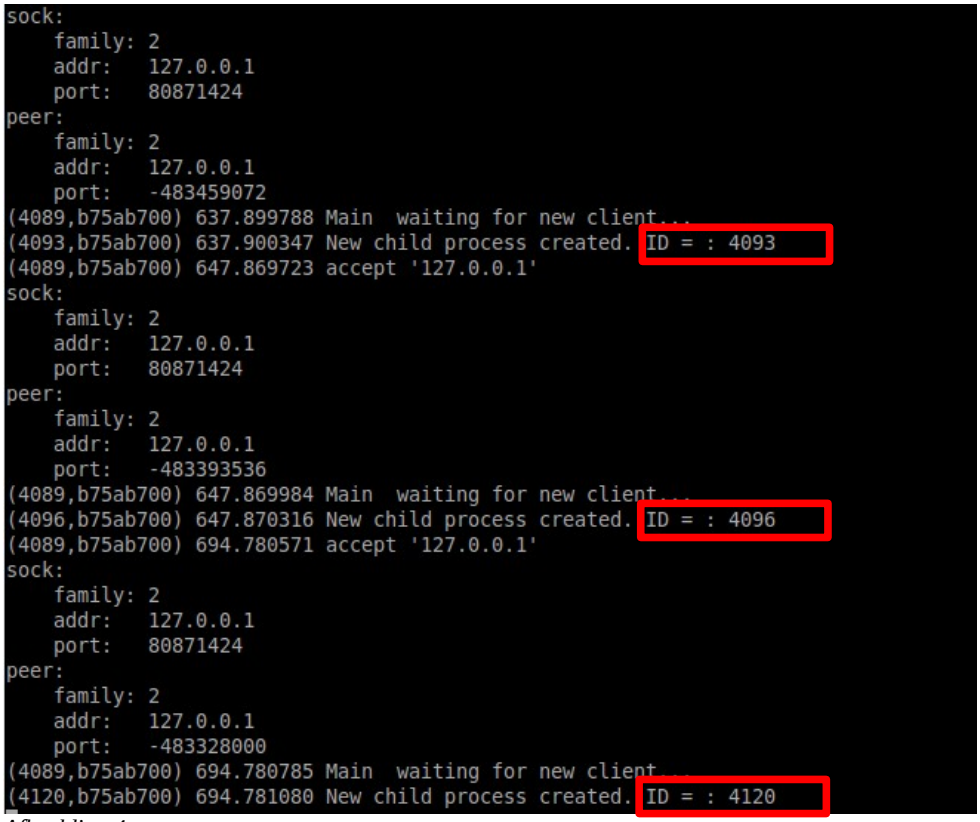
Uitwerking taak 3

Onderstaande code is aangemaakt in de *main* routine van 'TCPEchoServer-Fork.c'. In de *main* routine wordt gewacht op een TCPClient waarna er een *fork()* wordt uitgevoerd. Aan de hand van het verkregen *ProcesID* wordt eerst gecontroleerd of het resultaat van de *fork* instructie goed. Als het resultaat goed is wordt er gekeken of het nieuwe proces een *Child* of een *Parent* proces is. Een *Child* proces krijgt een Proces ID 0 en een *Parent* > 0. Als het een *Child process* is zal de communicatie met de cliënt worden afgehandeld. Een *Parent process* zal wachten op een nieuwe verbinding.

```
19
20      servSock = CreateTCPServerSocket (argv_port);
21
22      while (to_quit == false)               /* run until someone indicates to quit... */
23      {
24          clntSock = AcceptTCPConnection (servSock);
25
26          processID = fork();
27          if (processID < 0)
28          {
29              // fatal error, fork failed
30              DieWithError ("fork() failed");
31          }
32          else
33          {
34              if (processID == 0)
35              {
36                  // processID == 0: child process, handle client communication
37                  info_d ("New child process created. ID = ", getpid());
38                  HandleTCPClient (clntSock);
39                  // Child process terminates
40                  exit (0);
41                  info_d ("Child process stopped. ID = ", getpid());
42              }
43              else
44              {
45                  // processID > 0: main process
46                  info ("Main  waiting for new client...");
47              }
48          }
49      }
50
51      // server stops...
52      exit (0);
```

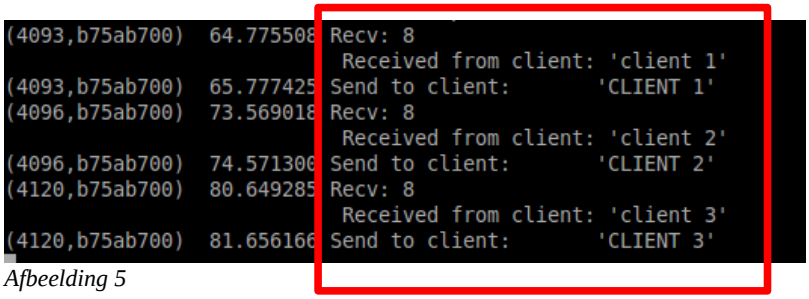
Resultaten

Als eerste wordt de TCPEchoServer-Fork opgestart waarna drie ChatClients verbinding maken met deze server. In afbeelding 4 is te zien dat de server voor ieder van de clients een eigen proces heeft aangemaakt met respectievelijk Proces ID 4093, 4096 en 4120.



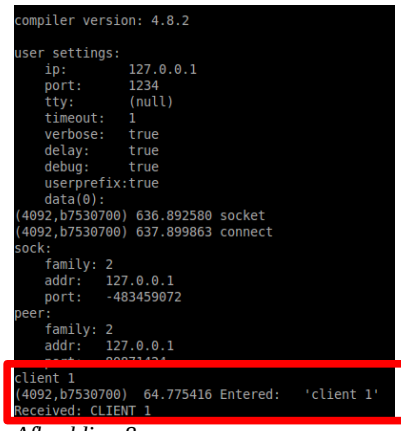
Afbeelding 4

Vervolgens zal vanaf ieder client (1, 2 en 3) de string 'client 1', 'client 2' en 'client 3' worden gestuurd. De server zal de string in hoofdletters terug sturen naar de betreffende client. Afbeelding 5 toont de server met de ontvangen string en wat er wordt terug gestuurd.

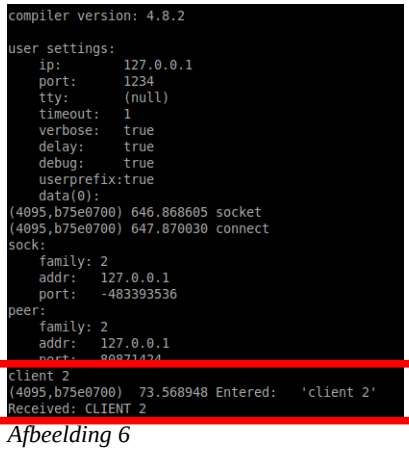


Afbeelding 5

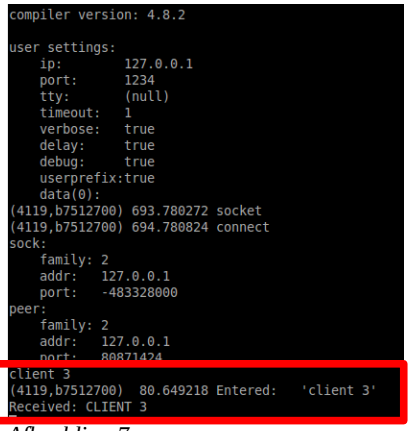
Onderstaande afbeelding 6, 7 en 8 tonen de clients met hun verstuurde en ontvangen data.



Afbeelding 8

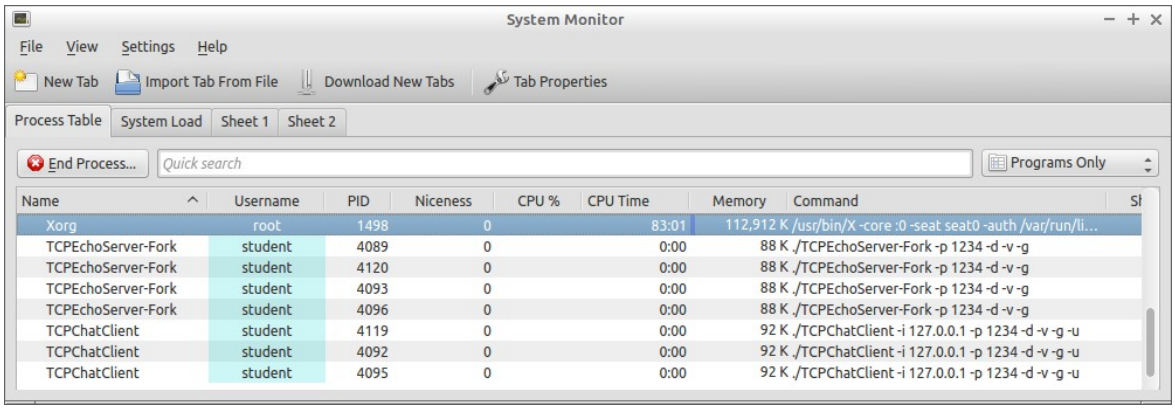


Afbeelding 6



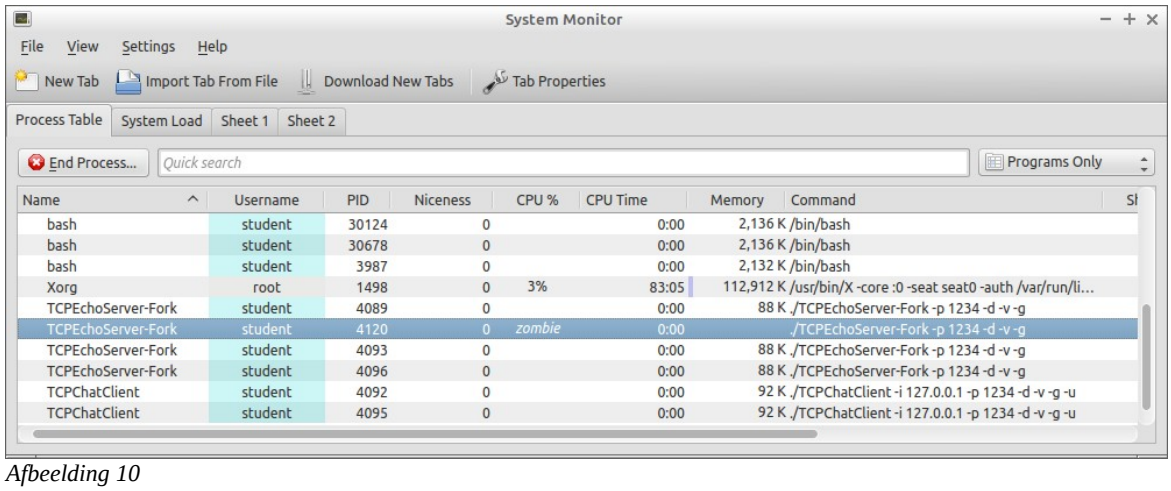
Afbeelding 7

Afbeelding 11 laat zien dat in deze situatie er vier server en drie cliënt processen actief zijn.



Afbeelding 9

Door een cliënt af te sluiten zal de bijbehorende geforkte Server de *Zombie state* krijgen. Afbeelding 10 toont dit.



Afbeelding 10

Uitwerking taak 4

Voor deze opdracht is het programma 'TCPEchoServer-Threads.c' compleet gemaakt. Als eerset zijn de volgende includes toegevoegd:

```
2   #include <stdlib.h>     // for exit()
3   #include <unistd.h>     // close()
```

De *main* routine is geïmplementeerd zoals te zien in afbeelding 11. In deze routine wordt nadat er een verbinding met een cliënt tot stand is gekomen, een nieuwe *thread* aangemaakt. Een van de parameters van de functie *pthread_create* is de clntSock.

```
12  int main (int argc, char *argv[])
13  {
14      int         servSock;     /* Socket descriptor for server */
15      int         clntSock;     /* Socket descriptor for client */
16      int         result;       /* Result for thread creation */
17      pthread_t   threadID;     /* Thread ID from pthread_create() */
18      bool        to_quit = false;
19
20      parse_args (argc, argv);
21
22      servSock = CreateTCPServerSocket (argv_port);
23
24      while (to_quit == false)               /* run until someone indicates to quit... */
25      {
26          clntSock = AcceptTCPConnection (servSock);
27
28          // Use the 'void *' parameter for passing clntSock
29          result = pthread_create (&threadID, NULL, myThread, (void *) clntSock);
30          if (result == 0)
31          {
32              info ("Succesfully created new thread");
33          }
34          else
35          {
36              DieWithError ("pthread_create()");
37          }
38      }
39
40      // server stops...
41      close (servSock);
42      exit (0);
43  }
```

Afbeelding 11

In afbeelding 12 is de implementatie te zien van de routine myThread. Hierin wordt de daadwerkelijke communicatie met de cliënt afgehandeld. m*yThread* leeft een eigen leven in een eigen *thread*. Iedere cliënt heeft dus feitelijk een eigen *myThread*. De threadArgs wordt gecast naar een integer.

```
45  static void *
46  myThread (void * threadArgs)
47  {
48      int clntSocket;
49
50      clntSocket = (int) threadArgs;
51
52      info ("Thread start");
53
54      if (pthread_detach (pthread_self ()) != 0)
55      {
56          DieWithError ("pthread_detach()");
57      }
58
59      // Do working
60      HandleTCPClient(clntSocket);
61
62      info ("Thread exit");
63      return (NULL);
64  }
```

Afbeelding 12

Compileren van de source

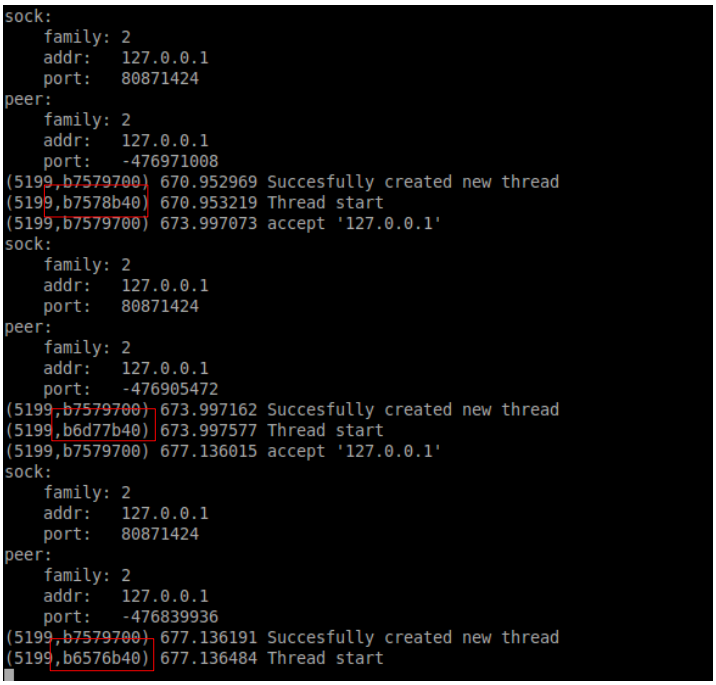
Om de source code te kunnen compileren is de Makefile aangepast zoals in afbeelding 13 te zien is. Met name de rood gemarkeerde delen zijn toegevoegd om te compileren met -lpthread libraries.

```
#
#   Makefile for IPC32
#
#   (c) Fontys 2010, Joris Geurts
#

BINARIES = TCPEchoClient TCPEchoServer TCPEchoServer-Fork TCPEchoServer-Thread TCPChatClient TCPChatServer

CC = gcc
CFLAGS = -g3 -Wall -W -Wshadow -Wcast-qual -Wwrite-strings
LDLIBS = -lpthread
CLIENT_AUX_OBJS = Auxiliary.o CreateTCPClientSocket.o
SERVER_AUX_OBJS = Auxiliary.o CreateTCPServerSocket.o AcceptTCPConnection.o HandleTCPClient.o
CHAT_CLIENT_AUX_OBJS = Auxiliary.o CreateTCPClientSocket.o
CHAT_SERVER_AUX_OBJS = Auxiliary.o CreateTCPServerSocket.o AcceptTCPConnection.o HandleTCPChatClient.o

all:    $(BINARIES)

clean:
    rm *.o $(BINARIES)

TCPEchoClient: TCPEchoClient.o $(CLIENT_AUX_OBJS)

TCPEchoServer: TCPEchoServer.o $(SERVER_AUX_OBJS)

TCPChatClient: TCPChatClient.o $(CLIENT_AUX_OBJS)

TCPChatServer: TCPChatServer.o $(CHAT_SERVER_AUX_OBJS)

TCPEchoServer-Fork: TCPEchoServer-Fork.o $(SERVER_AUX_OBJS)

TCPEchoServer-Thread: TCPEchoServer-Thread.o $(SERVER_AUX_OBJS) $(LDLIBS)

```

Afbeelding 13

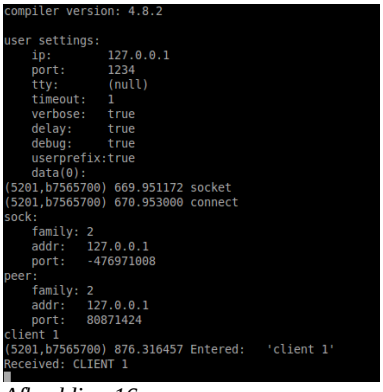
Resultaten

Afbeelding 14 toont de server verbonden met drie clients. De rode kaders laten zien dat iedere thread zijn eigen ID heeft.

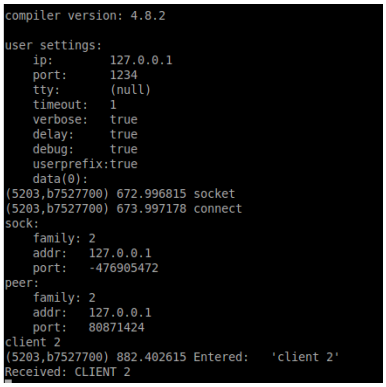


Afbeelding 14

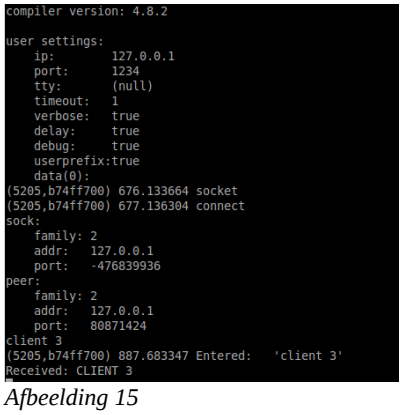
Afbeelding 15, 16 en 17 tonen alle drie de clients en hun verstuurde en ontvangen data.



Afbeelding 16

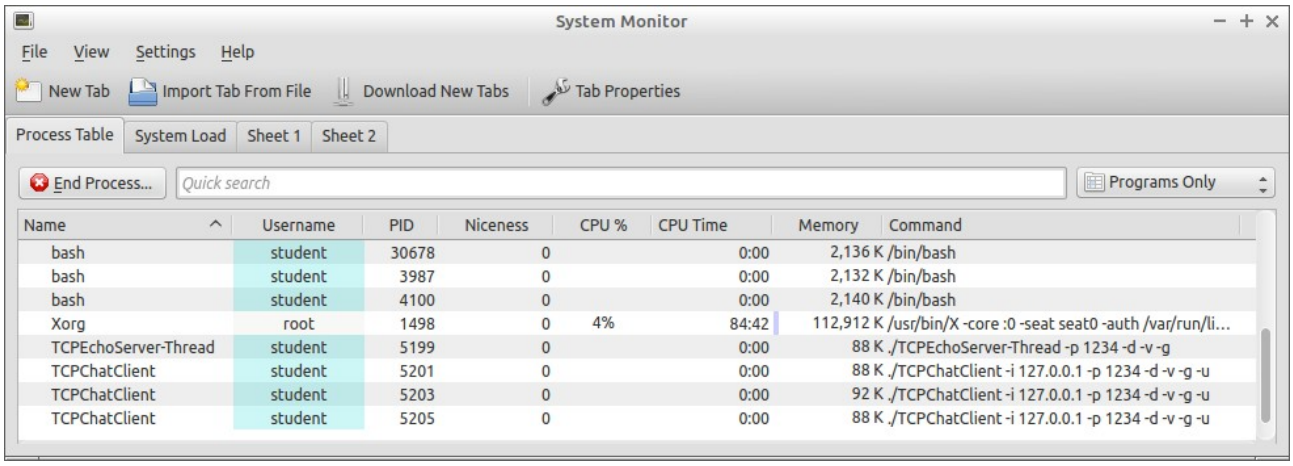


Afbeelding 17



Afbeelding 15

Afbeelding 18 toon dat er drie clients zijn en een server. Dit in tegenstelling tot het *forken* wat we eerst gedaan hebben.



Afbeelding 18

Afbeelding 19 laat zien dat alle drie de clients een voor een gesloten zijn en dat ook de threads netjes gestopt worden (Thread exit).

```
                        Received from client: 'Quit'
(5347,b75b2b40) 394.875571 Send to client:       'qUIT'
(5347,b75b2b40) 394.875650 recv: 0
(5347,b75b2b40) 394.875663 close
(5347,b75b2b40) 394.875669 Thread exit
(5347,b6db1b40) 399.909976 recv: 4
                        Received from client: 'Quit'
(5347,b6db1b40) 400.914346 Send to client:       'qUIT'
(5347,b6db1b40) 400.914413 recv: 0
(5347,b6db1b40) 400.914424 close
(5347,b6db1b40) 400.914453 Thread exit
(5347,b65b0b40) 403.959490 recv: 4
                        Received from client: 'Quit'
(5347,b65b0b40) 404.961878 Send to client:       'qUIT'
(5347,b65b0b40) 404.962097 recv: 0
(5347,b65b0b40) 404.962133 close
(5347,b65b0b40) 404.962140 Thread exit
```
Afbeelding 19

Uitwerking taak 5

Installatie WireShark. In eerste insantie de installatie via de Lubuntu Software Installer gedaan. Hiermee bleek een behoorlijk verouderde versie geïnstalleerd te zijn. Deze versie crashte zelfs bij het maken van een Flow Graph. Na enig speurwerk ben ik erachter gekomen dat ik het beste een versie uit een andere repository kon nemen. Deze heb ik als volgt geïnstalleerd:

```
$ sudo add-apt-repository ppa:dreibh/ppa
$ sudo apt-get update
$ sudo apt-get install wireshark
```

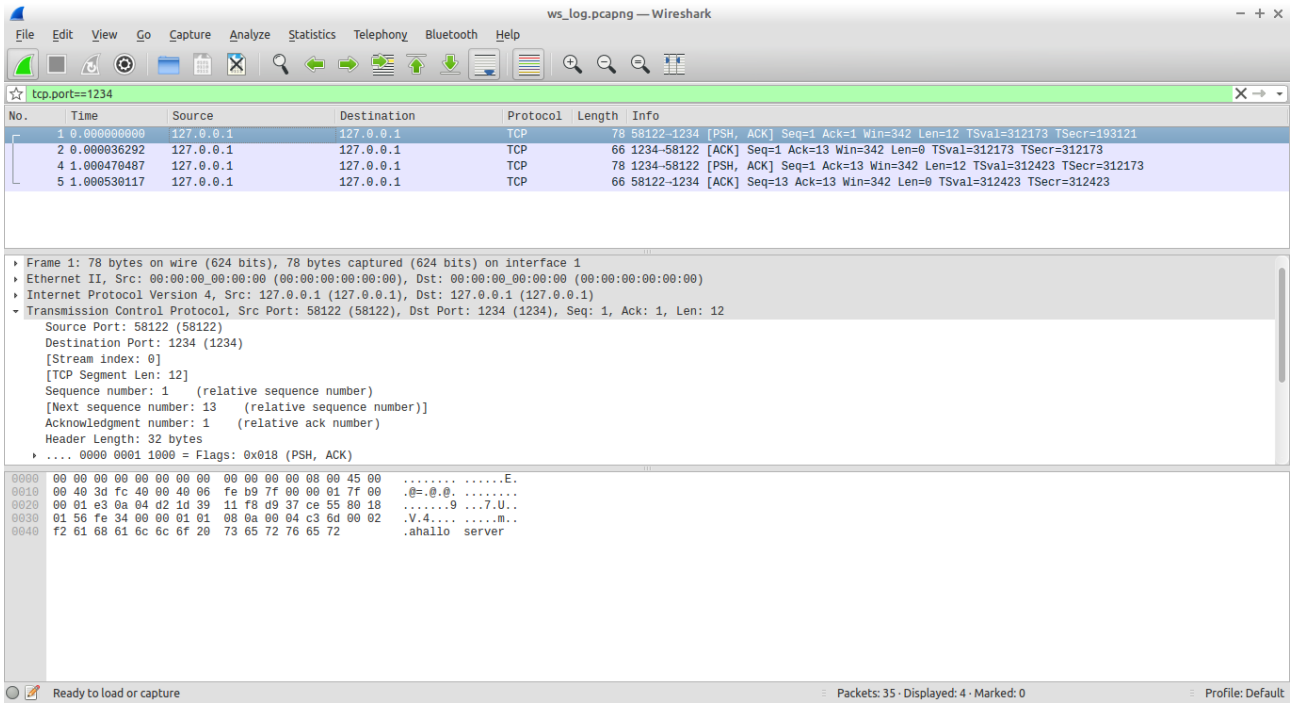
Na het bovenstaande had ik versie 1.99.6

Nu nog de toegang tot de netwerk interfaces. Als WireShark gestart wordt met sudo krijg ik nog steeds foutmeldingen. Ook hiervoor heb ik een oplossing gevonden. Door onderstaande opdrachten uit te voeren werkte alles naar behoren.

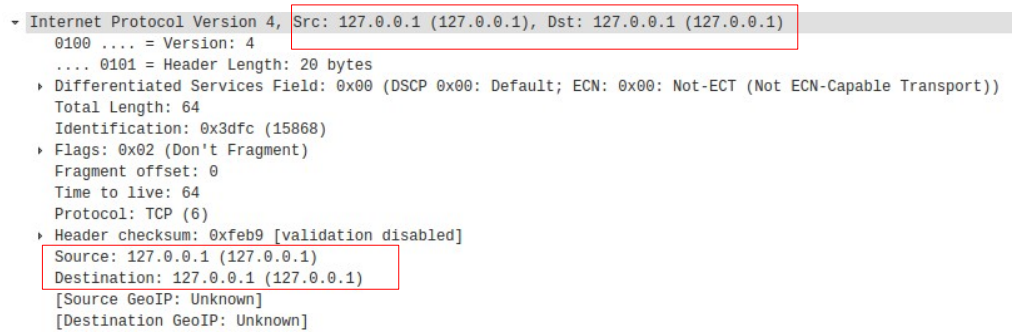
```
$ sudo dpkg-reconfigure wireshark-common
$ sudo usermod -a -G wireshark $USER
$ sudo reboot
```

IP-adressen, poort nummers en chat strings

Onderstaande afbeelding toont de complete WireShar interface. Hierna zal ik hiervan delen tonen om de gevraagde informatie aan te duiden.

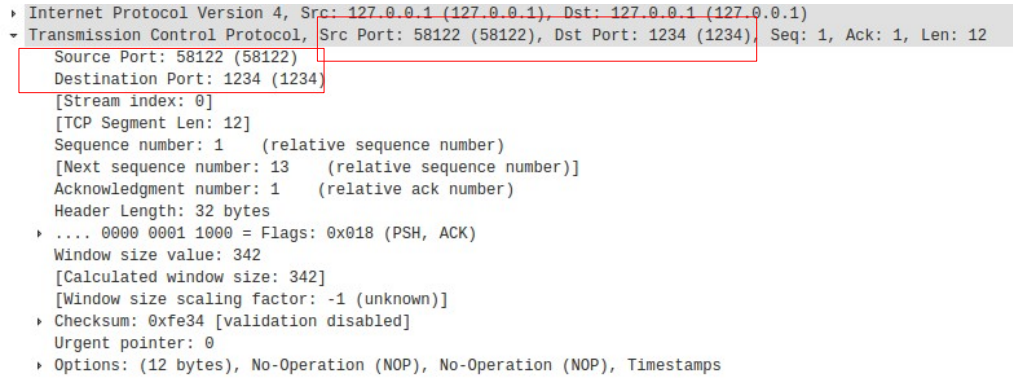


Het IP-adres kan gevonden worden in het midden gedeelte (Internet ProtocolVersion....) van het programma. Afbeelding 20 laat zien dat source en destination beide 127.0.0.1 zijn.



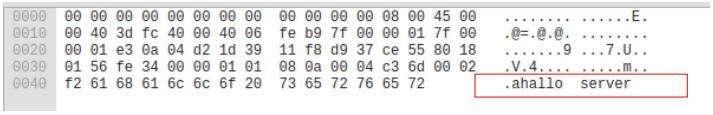
Afbeelding 20

Ook de poorten kunnen in dit gedeelte gevonden worden. Afbeelding 21 laat deze zien.

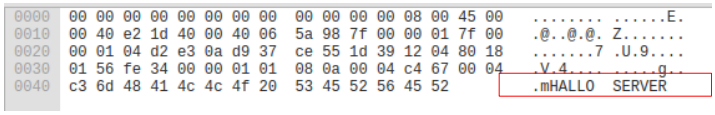


Afbeelding 21

De gezochte chat string kan gevonden worden in het onderste gedeelte van het programma. Afbeelding 22 laat het bericht zien dat de cliënt verstuurd. Afbeelding 23 laat het antwoord in hoofdletters van de server zien.



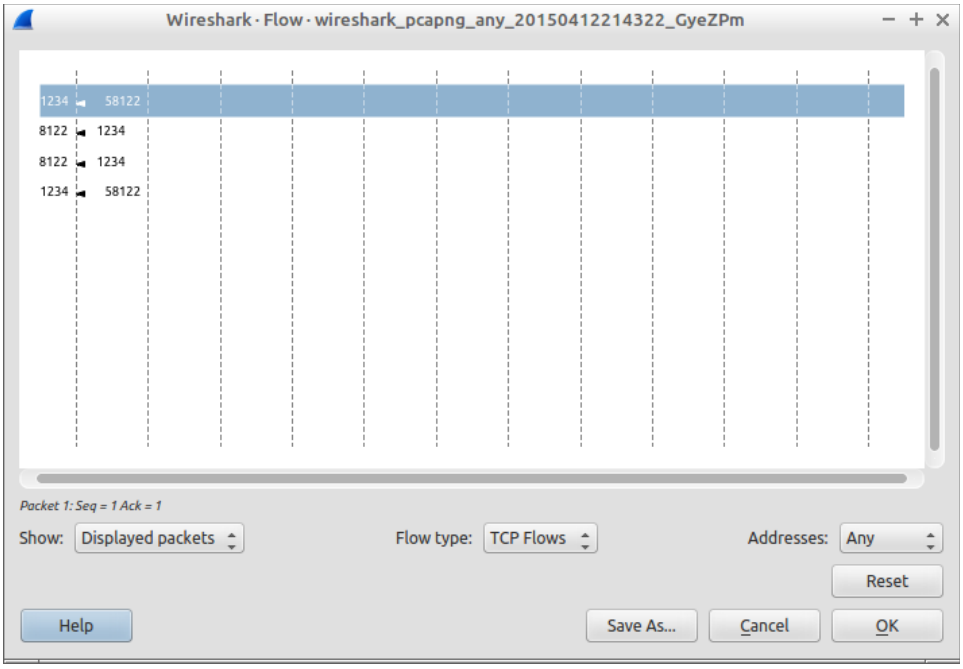
Afbeelding 22



Afbeelding 23

Flow Graph

Afbeelding 24 laat de Flow Graph zien met de gevraagde instellingen zien.



Afbeelding 24

De interface laat niet zo heel veel zien en er valt dus ook weinig over te schrijven Wel zijn de regels aan te klikken waardoor ook de packets in de Capture Window geselceteerd worden. De getallen die te zien zijn zijn de source en destination poort.