

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 schrijven 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 die 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 // TODO0: add code to print the received string; use printf()
30 add_nt(echoBuffer, recvMsgSize);
31 printf("Received from client: %s\n", echoBuffer);
32
33 // TODO0: add code to convert the upper/lower chars of the received string
34 invert_case(echoBuffer);

43
44 // TODO0: 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_nrofddata; 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.

```
student@student-fontys:~/bart/programming/school/opdracht.7/codes$ ./TCPEchoClient -i 127.0.0.1 -p 1234 -d -v -g aBcDeFgHiJkLmNoP 1234 XYZ
compiler version: 4.8.2

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'
(32626,b75d8700) 458.370946 socket
(32626,b75d8700) 459.382552 connect
sock:
  family: 2
  addr:   127.0.0.1
  port:   -531169280
peer:
  family: 2
  addr:   127.0.0.1
  port:   80871424
(32626,b75d8700) 460.396821 Sending to server: 'aBcDeFgHiJkLmNoP'
                        Received from server: 'AbCdEfGhIjKlMnOp'
(32626,b75d8700) 462.426869 Sending to server: '1234'
                        Received from server: '1234'
(32626,b75d8700) 464.454597 Sending to server: 'XYZ'
                        Received from server: 'xyz'
(32626,b75d8700) 466.483654 close & exit
student@student-fontys:~/bart/programming/school/opdracht.7/codes$
```

Afbeelding 2

```
student@student-fontys:~/bart/programming/school/opdracht.7/codes$ ./TCPEchoServer -p 1234 -d -v -g
compiler version: 4.8.2

user settings:
  ip:      (null)
  port:    1234
  tty:     (null)
  timeout: 1
  verbose: true
  delay:   true
  debug:   true
  userprefix: false
  data(0):
(32651,b75b6700) 612.393297 socket
(32651,b75b6700) 612.393355 bind: 1234
(32651,b75b6700) 612.393401 listen
(32651,b75b6700) 616.975321 accept '127.0.0.1'
sock:
  family: 2
  addr:   127.0.0.1
  port:   80871424
peer:
  family: 2
  addr:   127.0.0.1
  port:   -531103744
(32651,b75b6700) 617.976205 Recv: 16
                        Received from client: 'aBcDeFgHiJkLmNoP'
(32651,b75b6700) 618.977374 Send to client: 'AbCdEfGhIjKlMnOp'
(32651,b75b6700) 619.979825 recv: 4
                        Received from client: '1234'
(32651,b75b6700) 620.982209 Send to client: '1234'
(32651,b75b6700) 621.984564 recv: 3
                        Received from client: 'XYZ'
(32651,b75b6700) 622.986926 Send to client: 'xyz'
(32651,b75b6700) 623.989321 recv: 0
(32651,b75b6700) 623.989417 close
```

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 sock = CreateTCPClientSocket (argv_ip, argv_port);
22
23 while (strcmp (line, "Quit") != 0)
24 {
25     /* Send message to server */
26     fgets (line, sizeof (line), stdin);
27     remove_nl (line);
28     info_s ("Entered: ", line);
29
30     bytesSend = send (sock, line, strlen (line), 0);
31     if (bytesSend < 0)
32     {
33         DieWithError ("send() failed");
34     }
35
36     /* Receive message from server */
37     if (strcmp (line, "Quit") != 0)
38     {
39         bytesRcvd = recv (sock, echoBuffer, RCVBUFSIZE-1, 0);
40         if (bytesRcvd < 0)
41         {
42             DieWithError ("recv() failed");
43         }
44         if (bytesRcvd > 0)
45         {
46             add_nt (echoBuffer, bytesRcvd);
47             printf ("Received: %s\n", echoBuffer);
48         }
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     }
53 }
```

Resultaten

Met verschillende parameters

```
student@student-fontys:~/bart/programming/school/opdracht.7/code$ ./TCPChatClient -i 127.0.0.1 -p 1234 -d -v -g -u
compiler version: 4.8.2
user settings:
  ip:      127.0.0.1
  port:    1234
  tty:     (null)
  timeout: 1
  verbose: true
  delay:   true
  debug:   true
  userprefix:true
  data(0):
(2815,b7505700) 996.894432 socket
(2815,b7505700) 997.897109 connect
sock:
  family: 2
  addr:   127.0.0.1
  port:   -509607936
peer:
  family: 2
  addr:   127.0.0.1
  port:   80871424
Hallo daar
(2815,b7505700) 3.274141 Entered: 'Hallo daar'
Received: Hoe is het aan de andere kant?
Hier is het prima hoor
(2815,b7505700) 28.076546 Entered: 'Hier is het prima hoor'
Received: Goed om te horen
Ik ga er vandoor, doe!
(2815,b7505700) 52.629016 Entered: 'Ik ga er vandoor, doe!'
Received: Ok
Quit
(2815,b7505700) 61.078668 Entered: 'Quit'
(2815,b7505700) 62.081254 close & exit
student@student-fontys:~/bart/programming/school/opdracht.7/code$ ./TCPChatServer -p 1234
compiler version: 4.8.2
user settings:
  ip:      127.0.0.1
  port:    1234
  tty:     (null)
  timeout: 1
  verbose: true
  delay:   true
  debug:   true
  userprefix:true
  data(0):
(2815,b7505700) 996.894432 socket
(2815,b7505700) 997.897109 connect
sock:
  family: 2
  addr:   127.0.0.1
  port:   -509607936
peer:
  family: 2
  addr:   127.0.0.1
  port:   80871424
Hallo daar
(2815,b7505700) 3.274141 Entered: 'Hallo daar'
Received: Hoe is het aan de andere kant?
Hier is het prima hoor
(2815,b7505700) 28.076546 Entered: 'Hier is het prima hoor'
Received: Goed om te horen
Ik ga er vandoor, doe!
(2815,b7505700) 52.629016 Entered: 'Ik ga er vandoor, doe!'
Received: Ok
Quit
(2815,b7505700) 61.078668 Entered: 'Quit'
(2815,b7505700) 62.081254 close & exit
student@student-fontys:~/bart/programming/school/opdracht.7/code$
```

Zonder parameters

```
student@student-fontys:~/bart/programming/school/opdracht.7/code$ ./TCPChatClient -i 127.0.0.1 -p 1234
Hoi server
Received: Hallo client
Alles goed daar
Received: Ja hoor, alle prima hier
Ik ga stoppen
Received: Ok
Quit
student@student-fontys:~/bart/programming/school/opdracht.7/code$ ./TCPChatServer -p 1234
Received: Hoi server
Hallo client
Received: Alles goed daar
Ja hoor, alle prima hier
Received: Ik ga stoppen
Ok
Received: Quit
Quit
^C
student@student-fontys:~/bart/programming/school/opdracht.7/code$
```


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 *ProcessID* 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 *Process 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.

```
sock:
  family: 2
  addr: 127.0.0.1
  port: 80871424
peer:
  family: 2
  addr: 127.0.0.1
  port: -483459072
(4089,b75ab700) 637.899788 Main waiting for new client...
(4093,b75ab700) 637.900347 New child process created. ID = : 4093
(4089,b75ab700) 647.869723 accept '127.0.0.1'
sock:
  family: 2
  addr: 127.0.0.1
  port: 80871424
peer:
  family: 2
  addr: 127.0.0.1
  port: -483393536
(4089,b75ab700) 647.869984 Main waiting for new client...
(4096,b75ab700) 647.870316 New child process created. ID = : 4096
(4089,b75ab700) 694.780571 accept '127.0.0.1'
sock:
  family: 2
  addr: 127.0.0.1
  port: 80871424
peer:
  family: 2
  addr: 127.0.0.1
  port: -483328000
(4089,b75ab700) 694.780785 Main waiting for new client...
(4120,b75ab700) 694.781080 New child process created. ID = : 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.

```
(4093,b75ab700) 64.775508 Recv: 8
Received from client: 'client 1'
(4093,b75ab700) 65.777425 Send to client: 'CLIENT 1'
(4096,b75ab700) 73.569018 Recv: 8
Received from client: 'client 2'
(4096,b75ab700) 74.571306 Send to client: 'CLIENT 2'
(4120,b75ab700) 80.649285 Recv: 8
Received from client: 'client 3'
(4120,b75ab700) 81.656166 Send to client: 'CLIENT 3'
```

Afbeelding 5

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

```

compiler version: 4.8.2
user settings:
ip: 127.0.0.1
port: 1234
tty: (null)
timeout: 1
verbose: true
delay: true
debug: true
userprefix: true
data(0):
(4092,b7530700) 636.892580 socket
(4092,b7530700) 637.899863 connect
sock:
family: 2
addr: 127.0.0.1
port: -483459072
peer:
family: 2
addr: 127.0.0.1
port: 80871424
client 1
(4092,b7530700) 64.775416 Entered: 'client 1'
Received: CLIENT 1

```

Afbeelding 8

```

compiler version: 4.8.2
user settings:
ip: 127.0.0.1
port: 1234
tty: (null)
timeout: 1
verbose: true
delay: true
debug: true
userprefix: true
data(0):
(4095,b75e0700) 646.868605 socket
(4095,b75e0700) 647.870030 connect
sock:
family: 2
addr: 127.0.0.1
port: -483393536
peer:
family: 2
addr: 127.0.0.1
port: 80871424
client 2
(4095,b75e0700) 73.568948 Entered: 'client 2'
Received: CLIENT 2

```

Afbeelding 6

```

compiler version: 4.8.2
user settings:
ip: 127.0.0.1
port: 1234
tty: (null)
timeout: 1
verbose: true
delay: true
debug: true
userprefix: true
data(0):
(4119,b7512700) 693.780272 socket
(4119,b7512700) 694.780824 connect
sock:
family: 2
addr: 127.0.0.1
port: -483328000
peer:
family: 2
addr: 127.0.0.1
port: 80871424
client 3
(4119,b7512700) 80.649218 Entered: 'client 3'
Received: CLIENT 3

```

Afbeelding 7

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

Name	Username	PID	Niceness	CPU %	CPU Time	Memory	Command	SI
Xorg	root	1498	0		83:01	112,912 K	/usr/bin/X -core :0-seat seat0 -auth /var/run/li...	
TCPEchoServer-Fork	student	4089	0	0	0:00	88 K	/TCPEchoServer-Fork -p 1234 -d -v -g	
TCPEchoServer-Fork	student	4093	0	0	0:00	88 K	/TCPEchoServer-Fork -p 1234 -d -v -g	
TCPEchoServer-Fork	student	4096	0	0	0:00	88 K	/TCPEchoServer-Fork -p 1234 -d -v -g	
TCPChatClient	student	4119	0	0	0:00	92 K	/TCPChatClient -i 127.0.0.1 -p 1234 -d -v -g -u	
TCPChatClient	student	4092	0	0	0:00	92 K	/TCPChatClient -i 127.0.0.1 -p 1234 -d -v -g -u	
TCPChatClient	student	4095	0	0	0:00	92 K	/TCPChatClient -i 127.0.0.1 -p 1234 -d -v -g -u	

Afbeelding 9

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

Name	Username	PID	Niceness	CPU %	CPU Time	Memory	Command	SI
bash	student	30124	0		0:00	2,136 K	/bin/bash	
bash	student	30678	0		0:00	2,136 K	/bin/bash	
bash	student	3987	0		0:00	2,132 K	/bin/bash	
Xorg	root	1498	0	3%	83:05	112,912 K	/usr/bin/X -core :0-seat seat0 -auth /var/run/li...	
TCPEchoServer-Fork	student	4089	0		0:00	88 K	/TCPEchoServer-Fork -p 1234 -d -v -g	
TCPEchoServer-Fork	student	4120	0	zombie	0:00		/TCPEchoServer-Fork -p 1234 -d -v -g	
TCPEchoServer-Fork	student	4093	0		0:00	88 K	/TCPEchoServer-Fork -p 1234 -d -v -g	
TCPEchoServer-Fork	student	4096	0		0:00	88 K	/TCPEchoServer-Fork -p 1234 -d -v -g	
TCPChatClient	student	4092	0		0:00	92 K	/TCPChatClient -i 127.0.0.1 -p 1234 -d -v -g -u	
TCPChatClient	student	4095	0		0:00	92 K	/TCPChatClient -i 127.0.0.1 -p 1234 -d -v -g -u	

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 ("Successfully 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. *myThread* 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

Compilieren 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.

```
1 #
2 # Makefile for IPC32
3 #
4 # (c) Fontys 2010, Joris Geurts
5 #
6 |
7 BINARIES = TCPEchoClient TCPEchoServer TCPEchoServer-Fork TCPEchoServer-Thread TCPChatClient TCPChatServer
8
9 CC = gcc
10 CFLAGS = -g3 -Wall -W -Wshadow -Wcast-qual -Wwrite-strings
11 LDLIBS = -lpthread
12 CLIENT_AUX_OBJS = Auxiliary.o CreateTCPClientSocket.o
13 SERVER_AUX_OBJS = Auxiliary.o CreateTCPServerSocket.o AcceptTCPConnection.o HandleTCPClient.o
14 CHAT_CLIENT_AUX_OBJS = Auxiliary.o CreateTCPClientSocket.o
15 CHAT_SERVER_AUX_OBJS = Auxiliary.o CreateTCPServerSocket.o AcceptTCPConnection.o HandleTCPChatClient.o
16
17 all: $(BINARIES)
18
19 clean:
20 | rm *.o $(BINARIES)
21
22 TCPEchoClient: TCPEchoClient.o $(CLIENT_AUX_OBJS)
23
24 TCPEchoServer: TCPEchoServer.o $(SERVER_AUX_OBJS)
25
26 TCPChatClient: TCPChatClient.o $(CLIENT_AUX_OBJS)
27
28 TCPChatServer: TCPChatServer.o $(CHAT_SERVER_AUX_OBJS)
29
30 TCPEchoServer-Fork: TCPEchoServer-Fork.o $(SERVER_AUX_OBJS)
31
32 TCPEchoServer-Thread: TCPEchoServer-Thread.o $(SERVER_AUX_OBJS) $(LDLIBS)
33
34
```

Afbeelding 13

Resultaten

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

```
sock:
  family: 2
  addr: 127.0.0.1
  port: 80871424
peer:
  family: 2
  addr: 127.0.0.1
  port: -476971008
(5199,b7579700) 670.952969 Succesfully created new thread
(5199,b7578b40) 670.953219 Thread start
(5199,b7579700) 673.997073 accept '127.0.0.1'
sock:
  family: 2
  addr: 127.0.0.1
  port: 80871424
peer:
  family: 2
  addr: 127.0.0.1
  port: -476905472
(5199,b7579700) 673.997162 Succesfully created new thread
(5199,b6d77b40) 673.997577 Thread start
(5199,b7579700) 677.136015 accept '127.0.0.1'
sock:
  family: 2
  addr: 127.0.0.1
  port: 80871424
peer:
  family: 2
  addr: 127.0.0.1
  port: -476839936
(5199,b7579700) 677.136191 Succesfully created new thread
(5199,b6576b40) 677.136484 Thread start
```

Afbeelding 14

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

```
compiler version: 4.0.2
user settings:
  ip: 127.0.0.1
  port: 1234
  tty: (null)
  timeout: 1
  verbose: true
  delay: true
  debug: true
  userprefix: true
  data(0):
(5201,b7565700) 669.951172 socket
(5201,b7565700) 670.953000 connect
sock:
  family: 2
  addr: 127.0.0.1
  port: -476971008
peer:
  family: 2
  addr: 127.0.0.1
  port: 80871424
client 1
(5201,b7565700) 876.316457 Entered: 'client 1'
Received: CLIENT 1
```

Afbeelding 16

```
compiler version: 4.8.2
user settings:
  ip: 127.0.0.1
  port: 1234
  tty: (null)
  timeout: 1
  verbose: true
  delay: true
  debug: true
  userprefix: true
  data(0):
(5203,b7527700) 672.996815 socket
(5203,b7527700) 673.997178 connect
sock:
  family: 2
  addr: 127.0.0.1
  port: -476905472
peer:
  family: 2
  addr: 127.0.0.1
  port: 80871424
client 2
(5203,b7527700) 882.402615 Entered: 'client 2'
Received: CLIENT 2
```

Afbeelding 17

```
compiler version: 4.8.2
user settings:
  ip: 127.0.0.1
  port: 1234
  tty: (null)
  timeout: 1
  verbose: true
  delay: true
  debug: true
  userprefix: true
  data(0):
(5205,b74ff700) 676.133664 socket
(5205,b74ff700) 677.136304 connect
sock:
  family: 2
  addr: 127.0.0.1
  port: -476839936
peer:
  family: 2
  addr: 127.0.0.1
  port: 80871424
client 3
(5205,b74ff700) 887.683347 Entered: 'client 3'
Received: CLIENT 3
```

Afbeelding 15

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

Name	Username	PID	Niceness	CPU %	CPU Time	Memory	Command
bash	student	30678	0		0:00	2,136 K	/bin/bash
bash	student	3987	0		0:00	2,132 K	/bin/bash
bash	student	4100	0		0:00	2,140 K	/bin/bash
Xorg	root	1498	0	4%	84:42	112,912 K	/usr/bin/X -core :0 -seat seat0 -auth /var/run/li...
TCPEchoServer-Thread	student	5199	0		0:00	88 K	./TCPEchoServer-Thread -p 1234 -d -v -g
TCPChatClient	student	5201	0		0:00	88 K	./TCPChatClient -i 127.0.0.1 -p 1234 -d -v -g -u
TCPChatClient	student	5203	0		0:00	92 K	./TCPChatClient -i 127.0.0.1 -p 1234 -d -v -g -u
TCPChatClient	student	5205	0		0:00	88 K	./TCPChatClient -i 127.0.0.1 -p 1234 -d -v -g -u

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 instantie 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
```

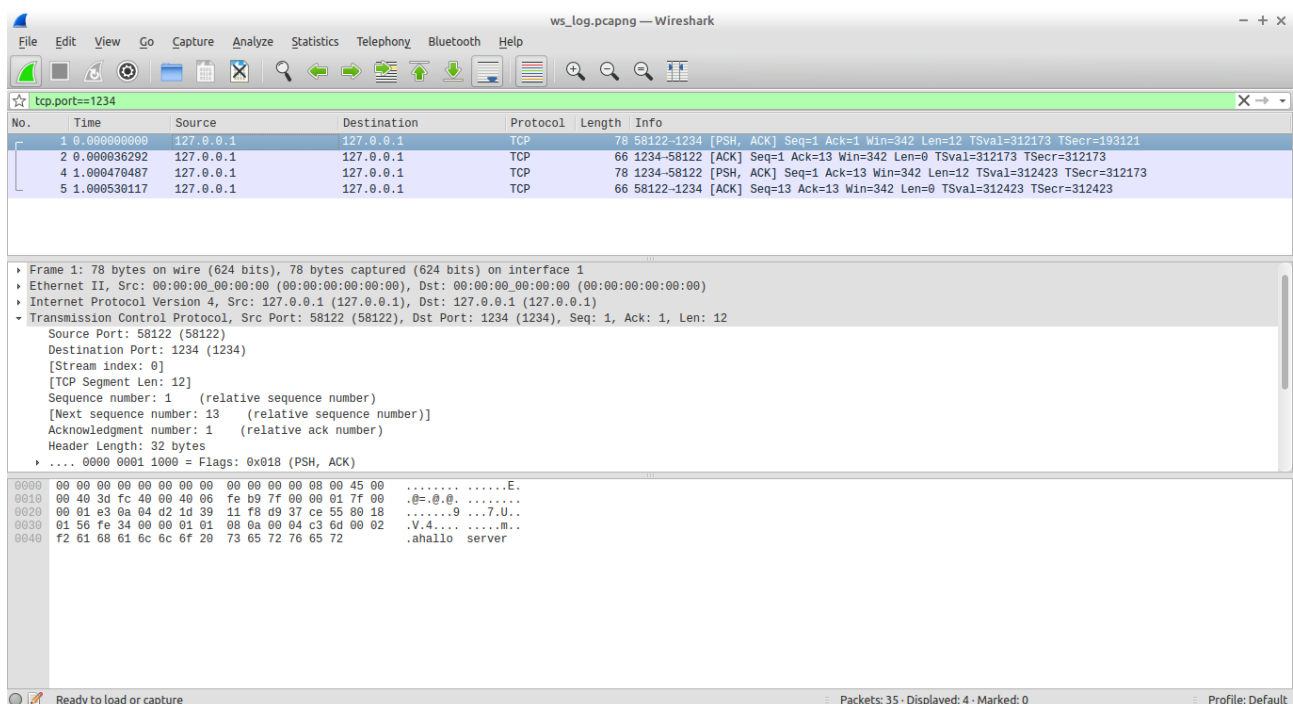
Na het bovenstaande had ik versie 1.99.6

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.

```

- Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes
  ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 64
  Identification: 0x3dfc (15868)
  ▶ Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  ▶ Header checksum: 0xfeb9 [validation disabled]
    Source: 127.0.0.1 (127.0.0.1)
    Destination: 127.0.0.1 (127.0.0.1)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]

```

Afbeelding 20

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

```

> Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
> Transmission Control Protocol, Src Port: 58122 (58122), Dst Port: 1234 (1234), Seq: 1, Ack: 1, Len: 12
  Source Port: 58122 (58122)
  Destination Port: 1234 (1234)
  [Stream index: 0]
  [TCP Segment Len: 12]
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 13 (relative sequence number)]
  Acknowledgment number: 1 (relative ack number)
  Header Length: 32 bytes
  > .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
  Window size value: 342
  [Calculated window size: 342]
  [Window size scaling factor: -1 (unknown)]
  > Checksum: 0xfe34 [validation disabled]
  Urgent pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps

```

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.

```
0000 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 .....E.
0010 00 40 3d fc 04 00 40 06 fe b9 7f 00 00 01 7f 00 @.=.@.
0020 00 01 e3 0a 04 d2 1d 31 f1 f8 d9 37 ce 55 00 18 .....9...7.U.
0030 01 56 fe 34 00 00 01 01 08 0a 00 04 c3 6d 00 02 .V.4.....m..
0040 f2 61 68 61 6c 6c 6f 20 73 65 72 76 65 72 .....ahallo server
```

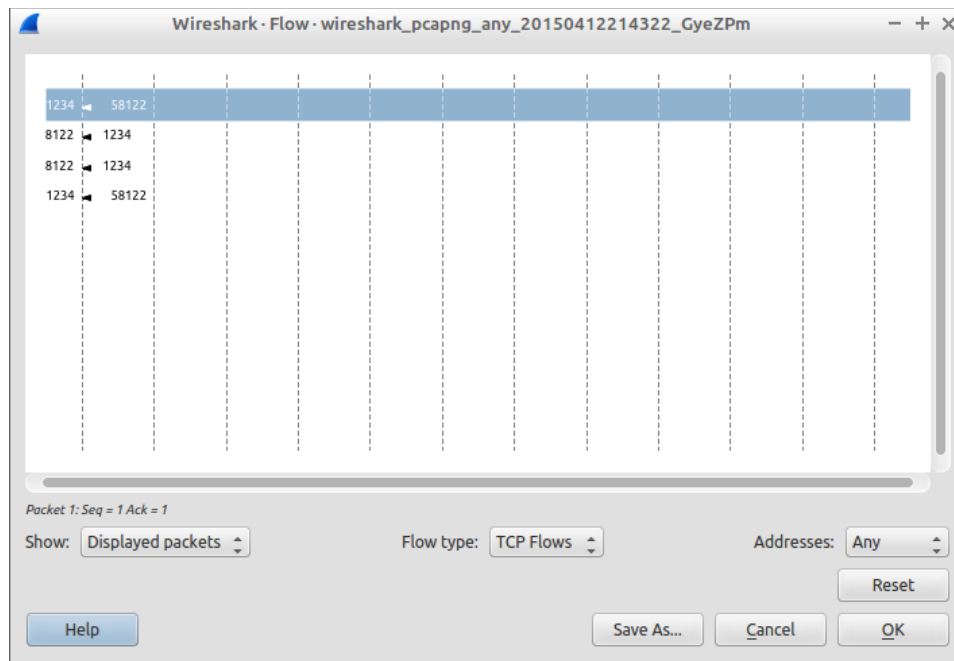
Afbeelding 22

```
0000 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 .....E.
0010 00 40 e2 1d 40 00 40 06 5a 98 7f 00 00 01 7f 00 .@. @. Z.....
0020 00 01 04 d2 e3 0a 09 37 ce 55 1d 39 12 04 80 18 .....7 .U.9....
0030 01 56 fe 34 00 00 01 08 0a 00 04 c4 67 00 04 .V.4.....g....
0040 c3 6d 48 41 4c 4c 4f 20 53 45 52 56 45 52 .....mHALLO SERVER
```

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 geselecteerd worden. De getallen die te zien zijn zijn de source en destination poort.