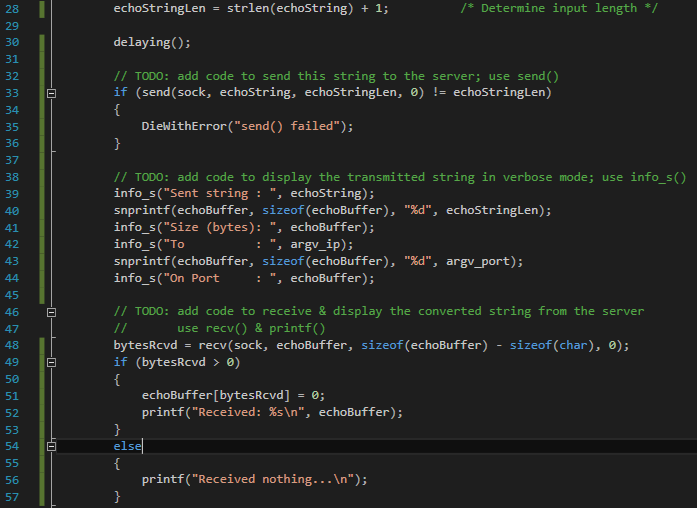
**Inleiding**

In deze opdracht wordt onderzocht hoe linux IPC werkt (het doel) d.m.v. een set C programma’s (het middel). De verwachting is dat twee of meerdere programma’s met elkaar kunnen communiceren (de verwachte uitkomst).

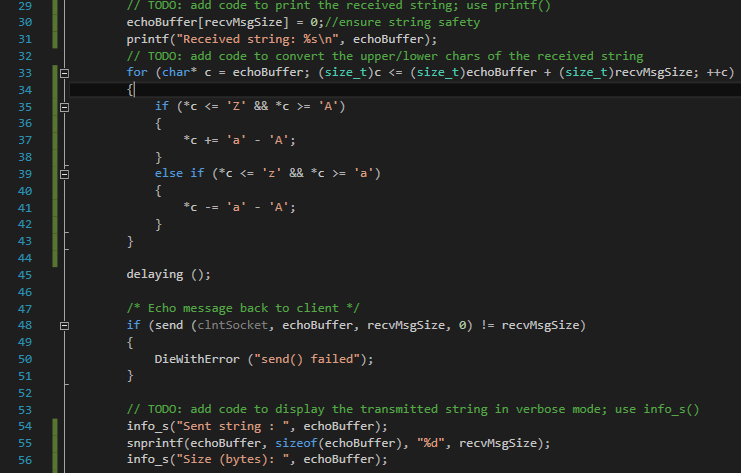
**Opdracht 1**

De aangepaste client (zoals volgens de opdracht en linux specificatie):

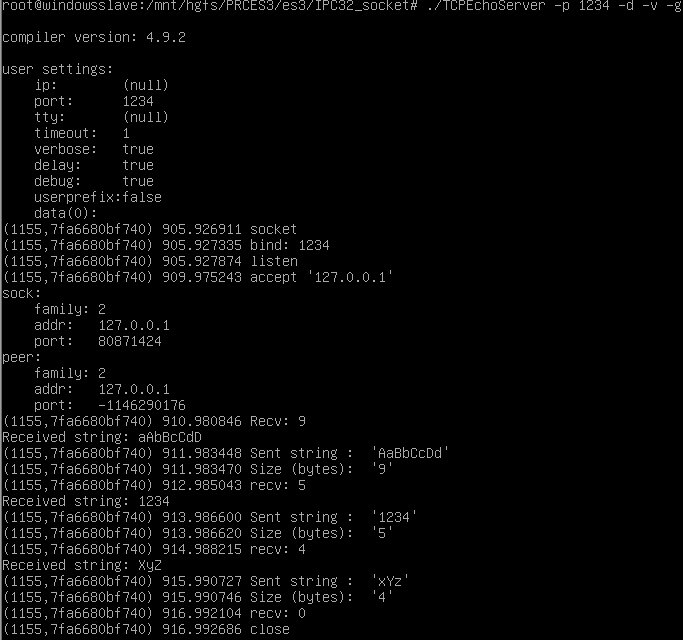


Voor null-string terminatie is de veiligheid van de applicatie mogelijk gecompromiseerd, daarom verander ik de code om de lengte op te vragen naar (+1).

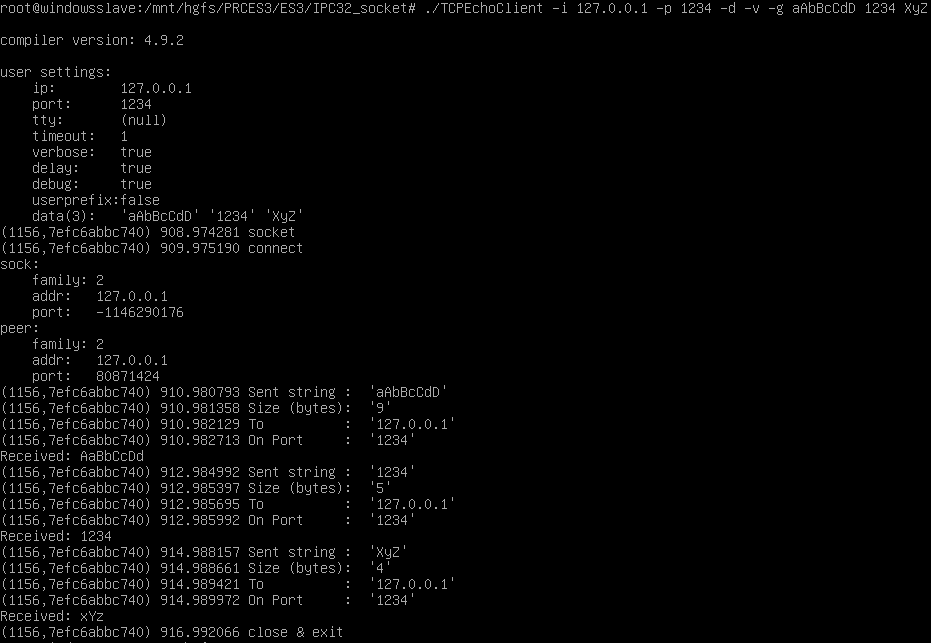
De aangepaste server (zoals volgens de opdracht en linux specificatie):



Terminal output van server:



Terminal output van client:



**Parameters**

De parameters:

-p <getal> is de keuze van de tcp communicatie poort

-v staat voor verbose mode enabled

-d voor enable delay

-i voor het ip waarmee de connectie (of bind tot stand moet worden gebracht bij de server)

-g zorgt ervoor dat debug mode aan staat

argc is het aantal argumenten dat aan het programma wordt gegeven (in main)

argv is een char\*\* array met de argumenten, hoeveelheid argumenten == argc (in main)

**Conclusie**

De server ontvangt de parameters die de echo client stuurt, ook ontvangt de client de response van de server, alles werkt naar behoren en zoals verwacht.

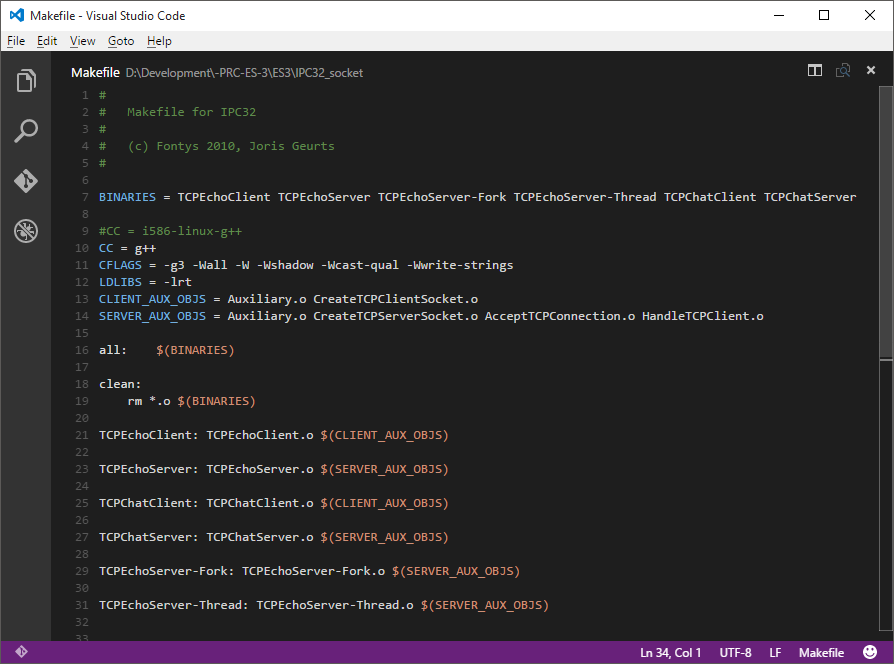
**Opdracht 2**

**Inleiding**

In deze opdracht wordt opdracht 1 uitgebreid naar een interactieve chat.

**Makefile**

Om de extra programma’s te compileren is het nodig om de makefile aan te passen:

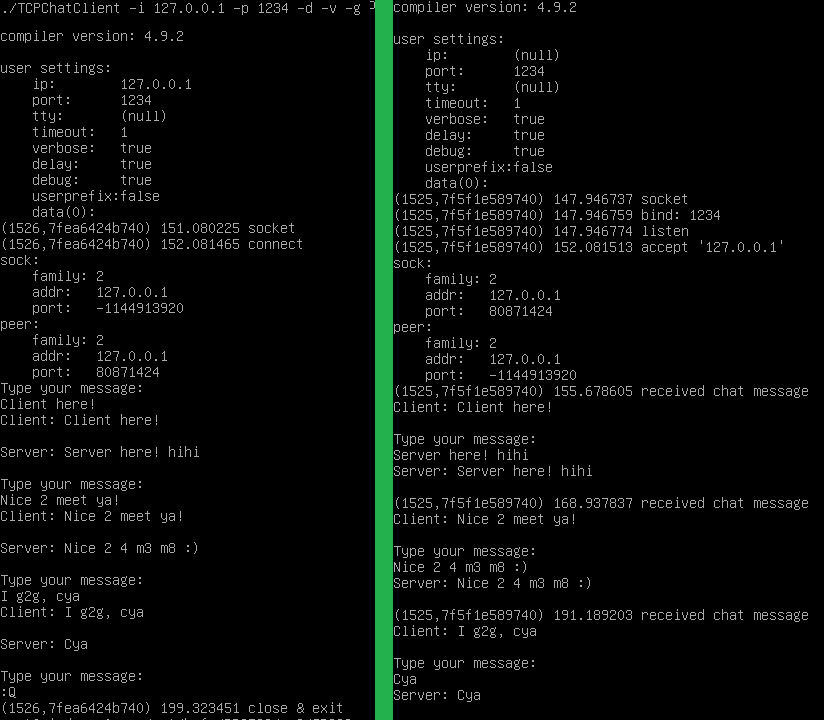


Hiermee zal de compilatie van de Chat programma’s mogelijk zijn via de meegeleverde makefile.

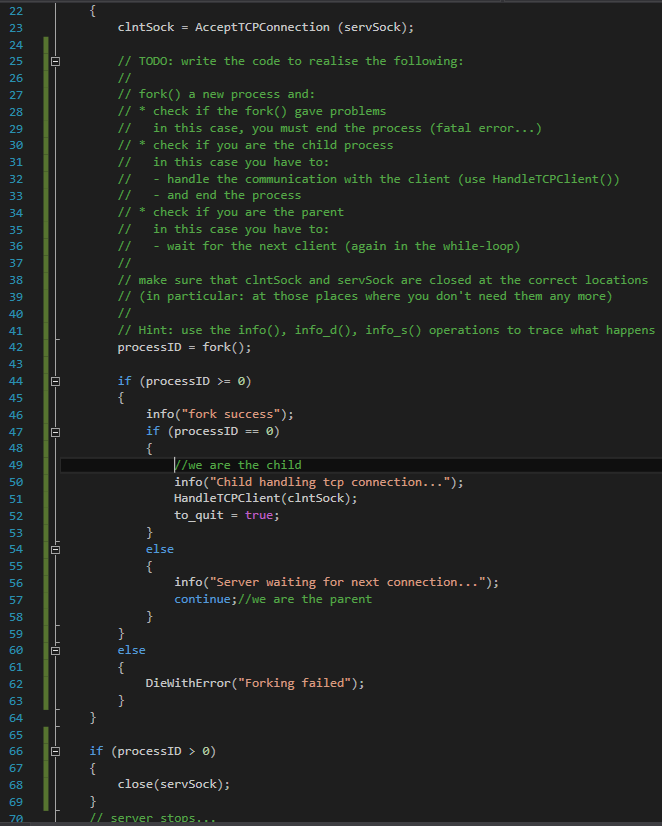
Zie de meegeleverde code voor veranderingen in o.a.:

* Auxiliary.h :: ANSIC\_getline prototype toegevoegd
* Auxiliary.c :: ANSIC\_getline definitie toegevoegd
* HandleTCPClient.h :: HandleTCPChatClient prototype toegevoegd
* HandleTCPClient.c :: HandleTCPChatClient definitie toegevoegd
* Nieuwe file: TCPChatClient.c
* Nieuwe file: TCPChatServer.c

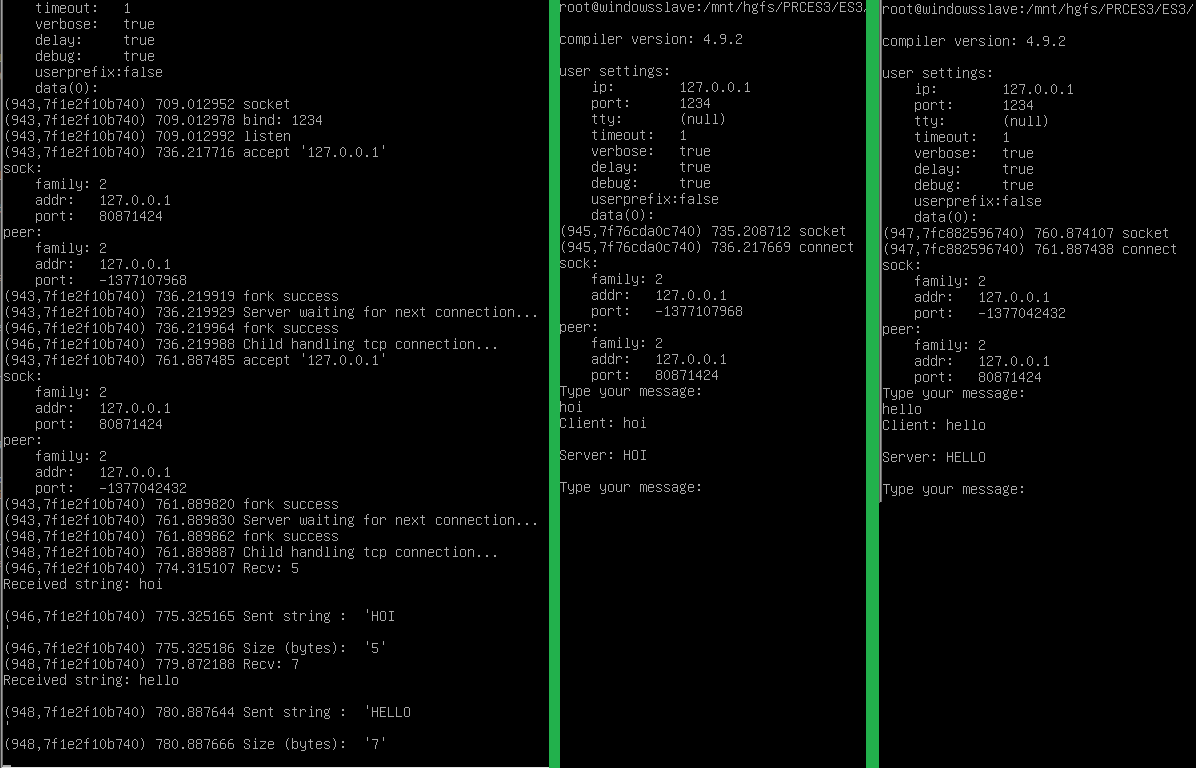
Na het opstarten van de TCPChatServer en TCPChatClient heeft de volgende conversatie plaatsgevonden (:Q == close):



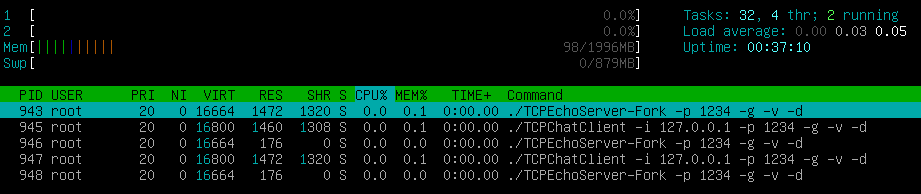
Hierna is de TCPEchoServer-Fork aangepast en twee TCPChatClient’s gebruikt om de werking te testen:



De test:



Zo als te zien is werkt de server naar verwachting, er worden inderdaad extra child processen gespawned voor elke client:



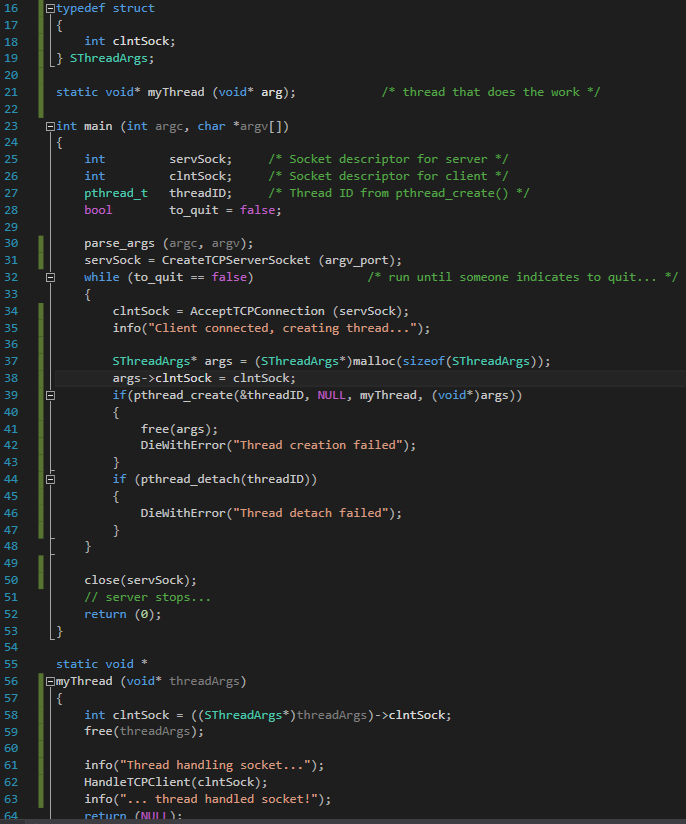
Voor het gebruik van threads zijn de volgende aanpassingen gedaan:

In de makefile de LD\_LIBS naar:

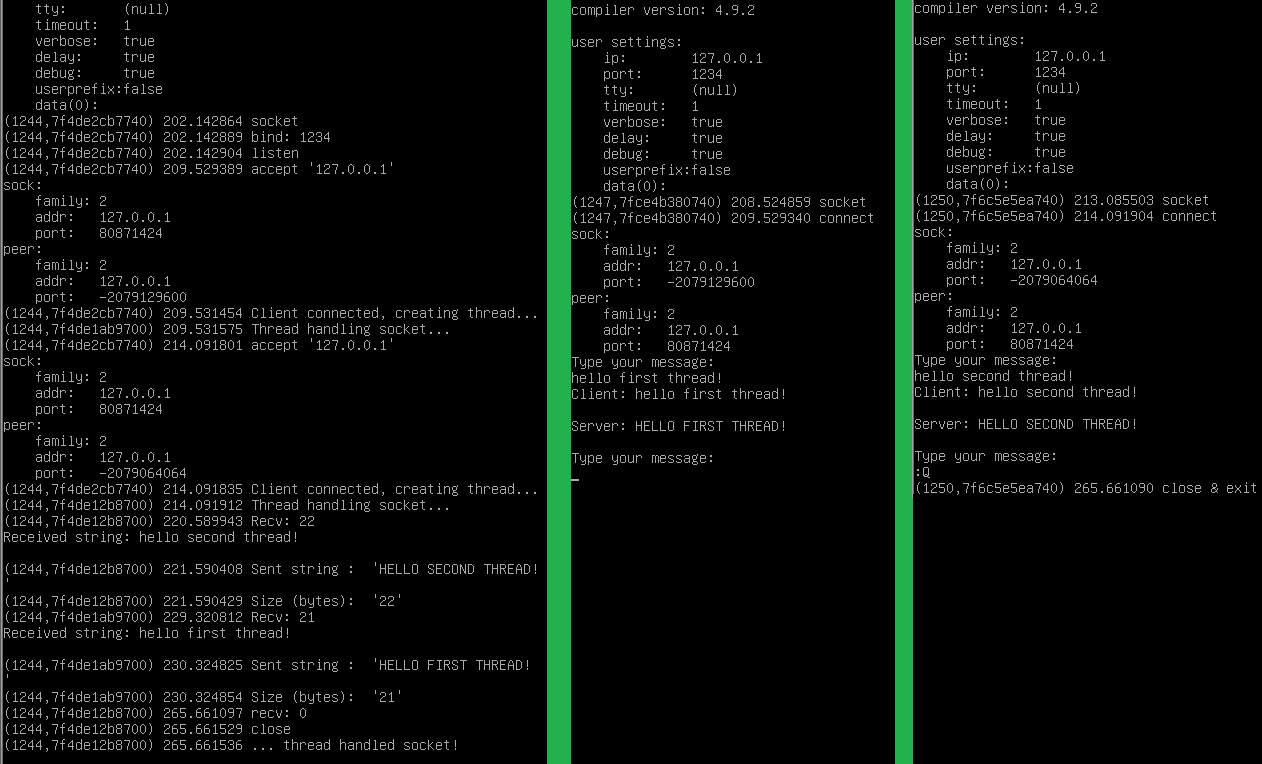
LDLIBS = -lrt –lpthread

Veranderd, zodat pthread meegelinkt wordt na compilatie.

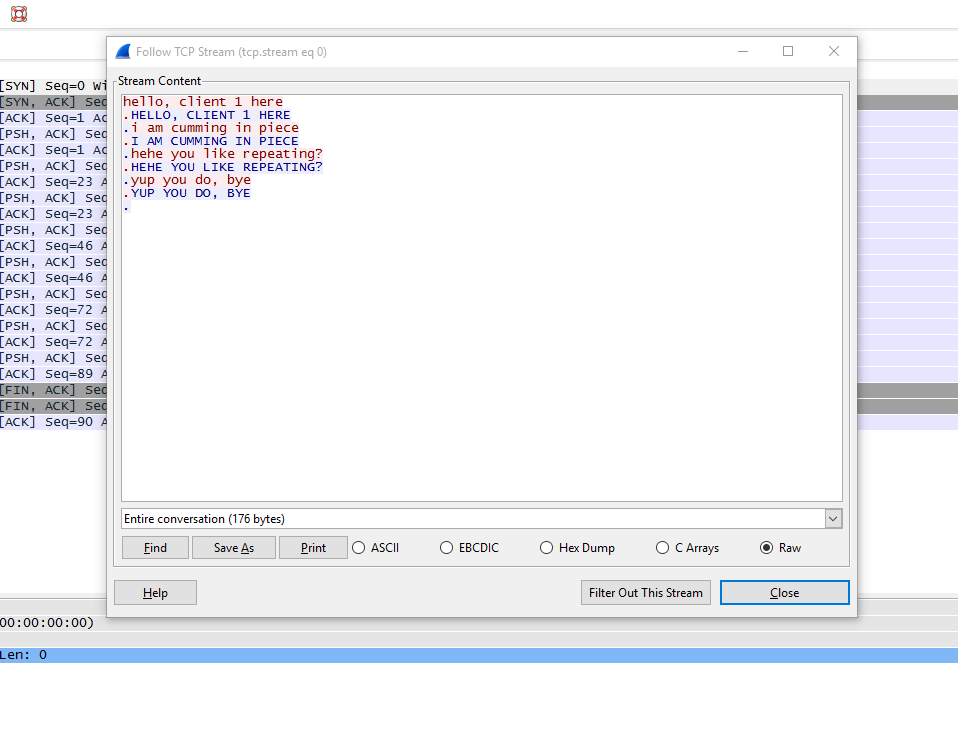
De code in the thread server:



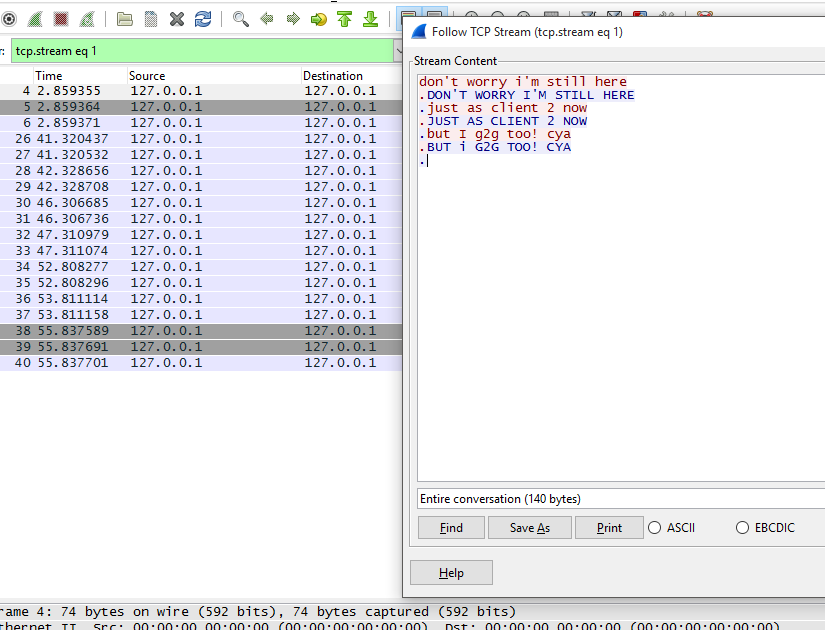
De test van de threaded server met TCPChatClient’s:



Met tcpdump –i lo ‘port 1234’ –w dump.dat worden de paketten op hetl inux systeem tussen de gemaakte processen opgeslagen om later in WireShark te bekijken, zo kun je rechts klikken op een packet en follow tcp stream toepassen, met het volgende als resultaat:

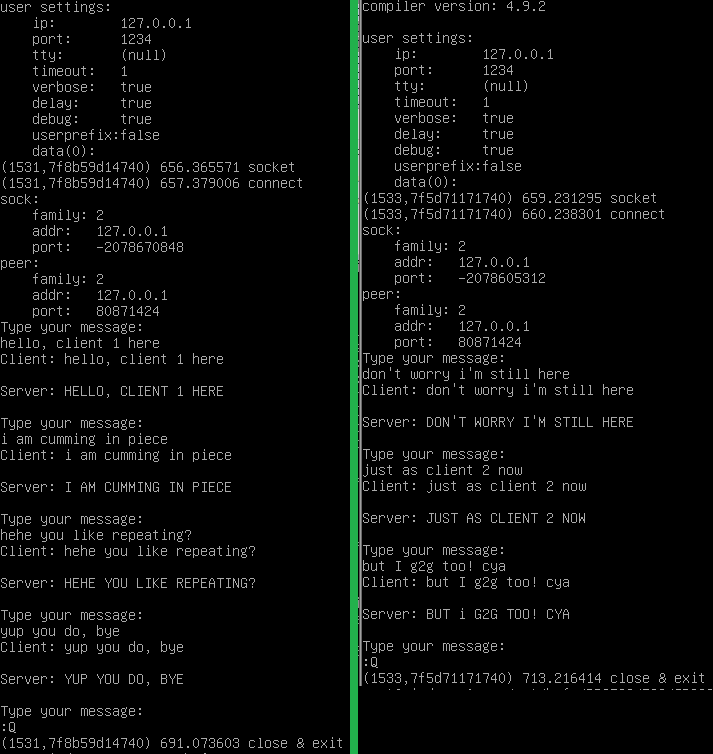


Client 1 (tcp stream 0)

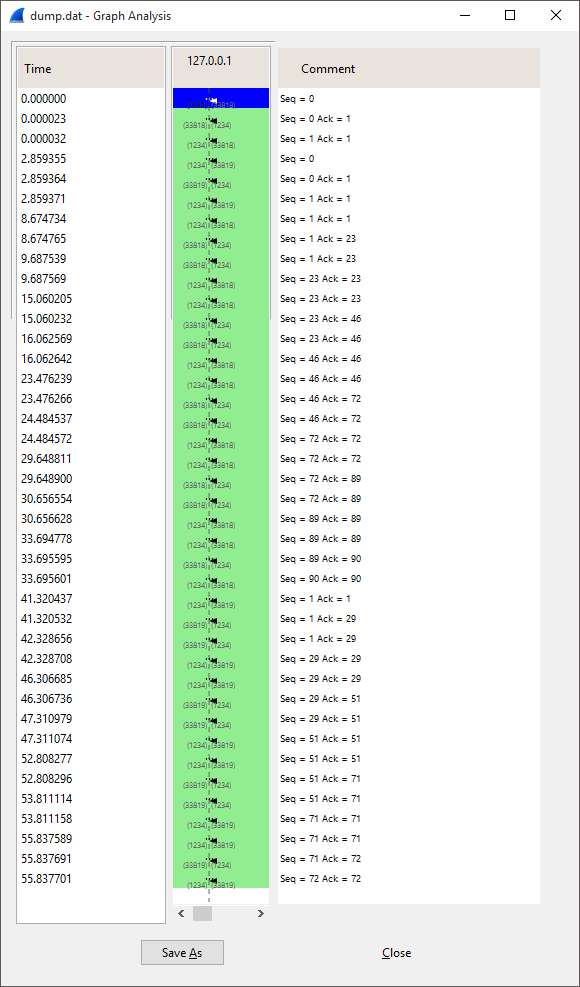


Client 2 (tcp stream 1)

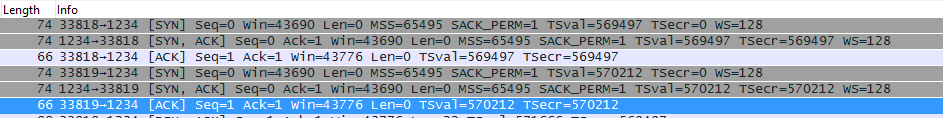
De streams komen overeen met de gevoerde ‘conversaties’:



In wireshark kan ook een statediagram opgevraagd worden, dit is het resultaat:



Dit is de uitwisseling van de pakketten en acknowledgements. Er kan worden gezien dat de verbinding tot stand komt door de eerste 6 pakketten (voor client 1):



Dus een sequentie van SYN, SYN, ACK, ACK, SYN, SYN, ACK, ACK is een ‘handshake’ en mechanisme om een tcp stream tot stand te brengen. Daarna worden de raw bytes gestuurd over de stream voor de uitwisseling van de data (maar de packets zijn door verschillende netwerk layers ingepakt in gestructureerde data pakketjes).