ES3/OS opdracht – Forking, Threads en Sockets

In deze opdracht gaan we dieper in op Client / Server communicatie in (fysiek) gedistribueerde systemen. De geijkte manier van communiceren is dan via het TCP/IP protoco gebruik makend van de *socket* APl. Om meerdere Clients tegelijk te kunnen bedienen kan de Server gebruik maken van *forking* of *multi-threading*.

# Voorbereidingen

Bestudeer Stallings paragrafen 4.1, 4.2, 4.6 en hoofdstuk 17 (Networking), welke laatste online beschikbaar is op: <https://www.box.com/shared/mbh0v0f6nx>

# Taken

## Echo client/server

Pak het bestand socket.zip uit. Dit bevat 2 programma's, een client TCPEchoClient en een server TCPEchoServer, maar de programma's zijn nog niet compleet. (Er zitten nog andere programma's in de zip, maar die zijn voor latere opdrachten)

De bedoeling is: een client maakt verbinding met een server, de client stuurt data naar de server. De server stuurt de data weer terug. Echter: de server zet kleine letters om in hoofdletters en zet grote letters om in kleine letters.

De source files kan je als volgt compileren:

make

Hierbij wordt gebruik gemaakt van de Makefile die in dezelfde directory staat.

Als je de onderstaande uitbreidingen gemaakt hebt, en als het compileren gelukt is, dan kan je de executables als volgt opstarten (in twee aparte shells):

./TCPEchoServer -p 1234 -d -v -g

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

* Wat betekenen de parameters?  
  (als je nog niet thuis bent in argc en argv, kijk dan op <http://publications.gbdirect.co.uk/c_book/chapter10/arguments_to_main.html> )

Maak de volgende uitbreidingen (zoek in de code naar comments met de tekst "TODO"):

* de server (functie HandleTCPClient())
  + een ontvangen string wordt met '\0' afgesloten
  + alle data die ontvangen wordt, wordt op de terminal afgedrukt
  + alle ontvangen hoofdletters worden kleine letters en omgekeerd, en dan wordt het teruggezonden
  + in de verbose mode wordt alle verzonden data ook op de terminal afgedrukt
* de client (file TCPEchoClient.c, functie main())
  + verzend alle strings van de command line één voor één naar de server en wacht op het antwoord van de server
  + een ontvangen string wordt met '\0' afgesloten
  + alle data die ontvangen wordt, wordt op de terminal afgedrukt
  + in de verbose mode wordt alle verzonden data ook op de terminal afgedrukt

## Chat client/server

Maak kopieën van TCPEchoClient.c en TCPEchoServer.c (noem ze bijvoorbeeld TCPChatClient.c en TCPChatServer.c ) en maak daarvan twee nieuwe programma's TCPChatClient en TCPChatServer om te kunnen chatten tussen een client en een server.

De client haalt zijn data *niet* uit de command line, maar van de terminal

Als een server data ontvangt van de client, dan stuurt de server dit *niet* terug naar de client, maar de server wacht dan op data van de terminal en verzendt dat naar de client.

De chat mag nog eenvoudig zijn: eerst wacht de client op een regel van de gebruiker, en als die regel verzonden is (en aangekomen bij de server), dan wacht de server op een regel van de gebruiker, etc.

Verzin een speciale regel (bijvoorbeeld: 'Quit') om de chat te beëindigen.

Compileren: maak een uitbreiding in Makefile om de nieuwe executables te bouwen.

Tip: je hoeft niet *beide* programma's helemaal af te ronden voordat je kunt testen: TCPChatClient kan ook met een TCPEchoServer praten, en TCPChatServer ook met een TCPEchoClient.

## Echo client/server fork

In het programma fork.c zie je hoe een proces zichzelf kan fork-en, en hoe de twee processen een eigen leven gaan leiden.

Voltooi het programma TCPEchoServer-Fork (die staat in de zip).

Voor elke client die zich aanmeldt start deze server een nieuw *proces* op die de communicatie met de client afhandelt (en daarna stopt het nieuwe proces).

Je kunt uiteindelijk meerdere TCPEchoClient's kunnen opstarten (elk in een eigen shell) die tegelijkertijd met deze server communiceren (gebruik de optie -d om de activiteiten niet te snel te laten verlopen).

Toon met screenshots aan dat de server meerdere clients tegelijkertijd kan bedienen.

## Echo client/server threads

In het programma thread.c zie je hoe threads gebruikt kunnen worden.

Voltooi het programma TCPEchoServer-Thread (die staat in de zip).

Voor elke client die zich aanmeldt start deze server een nieuwe *thread* op die de communicatie met de client afhandelt (en daarna stopt de thread).

Je kunt uiteindelijk meerdere TCPEchoClient's kunnen opstarten (elk in een eigen shell) die tegelijkertijd met deze server communiceren (gebruik de optie -d om de activiteiten niet te snel te laten verlopen).

Toon met screenshots aan dat de server meerdere clients tegelijkertijd kan bedienen.

# Opleveren

Je dient een document op te leveren waarin je de bovenstaande experimenten uitwerkt. Voeg, waar nodig, screendumps toe om een en andere duidelijk te maken. Lever ook de bijbehorende source code in van de programma’s die je gemaakt of aangepast hebt.