NS3 Lab 2 – TCP Client/Server Programming in C

Dr Colin Perkins
School of Computing Science
University of Glasgow
http://csperkins.org/teaching/ns3/

16 January 2013

Introduction

The laboratory sessions for Networked Systems 3 (NS3) will introduce you to network programming in C on Unix/Linux systems. There are weekly labs for this course, during which you will complete several exercises. These exercises will build on your knowledge of C programming and pthreads from the Advanced Programming 3 course last semester, and on the material in the NS3 lectures. There are a mixture of formative and summative exercises. The formative exercises are intended to give you practice in programming networked systems in C; they are not assessed. The two summative exercises are assessed, and are worth a total of 20% of the marks for this course.

This is NS3 lab 2, a further introduction to TCP client/server programming in C. It comprises three formative exercises, and should be completed during the timetabled laboratory session in week 2 of the semester, and outside the lab over the following week. It it expected that these exercises will take a total of around 4 hours to complete, with Formative Exercise 2 comprising the majority of the coding time (Formative Exercises 3 and 4 should require only very minor changes to the code written for Formative Exercise 2, if you have structured your code correctly). This work is not assessed, but is important preparation for the summative exercises later in the course.

Formative Exercise 2: TCP Date/Time Server

The second formative exercise introduces you to simple application-layer network protocols. You should write three programs:

`dt_server` The server should listen on TCP port 5001 for incoming connections. It should accept a connection, read and process a request, and send a response to the client. There are two types of request that can be received:

DATE On receiving the text "DATE" (without the quotes) followed by a carriage return and new line ("\r\n"), the server should send the current date back to the client, followed by a carriage return and new line. The standard library functions declared in the `<time.h>` header can be used to get and format the date.

TIME On receiving the text "TIME" (without the quotes) followed by a carriage return and new line ("\r\n"), the server should send the current time back to the client, followed by a carriage return and new line. The standard library functions declared in the `<time.h>` header can be used to get and format the time.

After sending the response to the client, the server should close the connection, and exit.

d_client This client should connect to TCP port 5001 of a host named on the command line, and send the text "DATE\r\n" (without the quotes). After sending the request, the client should read and print the response, close the connection, and exit.

t_client This client should connect to TCP port 5001 of a host named on the command line, and send the text "TIME\r\n" (without the quotes). After sending the request, the client should read and print the response, close the connection, and exit.

Run your server, and demonstrate that it correctly responds to requests from the two clients.

As before, you are *required* to write a simple Makefile to compile your code, rather than running the compiler by hand. You are also *strongly advised* to enable all compiler warnings (at *minimum*, use `gcc -W -Wall -Werror`), and to fix your code so it compiles without warnings. Compiler warnings highlight code which is legal, but almost certainly doesn't do what you think it does. Use them to help you find problems.

Formative Exercise 3: Handling Multiple Requests

A server that exits after processing a single request is not especially useful. Extend the server you wrote in Formative Exercise 2, so that it loops, processing multiple requests from the same connection, until the client closes the connection.

Modify the `d_client` and `t_client` programs you wrote in Exercise 2 so that rather than exit after sending one request, they loop sending repeated requests to the server on the same connection. Each request should be sent a few seconds after the previous, and each response from the server should be printed immediately it is received. Once they have sent 10 requests, and printed 10 responses, these clients should close the connection, and exit.

Test your extended `dt_server` with each client in turn, to demonstrate that it can processes multiple requests.

Formative Exercise 4: A Multi-threaded Server

There are two ways of handling multiple simultaneous connections to a server: the server may synchronously multiplex several connections using the `select()` system call, or it may asynchronously manage the connections using multiple threads. In this Formative Exercise, you will build a server that multiplexes several connections using threads, since this is the most scalable approach for multicore systems.

Write a program `t_dt_server`, based on the `dt_server` that you wrote for Formative Exercise 3. This new server should start a thread for each connection it accepts, and pass the file descriptor for the connection to that thread. Each thread should process the requests for the given connection, and exit when the connection is closed. The server should keep listening on the original socket, accepting an arbitrary number of new connections.

Test your server by allowing running severals clients at once, using a mixture of `d_client` and `t_client`, all connected to the same server. Demonstrate that your server correctly responds to requests from each client, and that it doesn't mix up responses to different requests. Demonstrate what happens when large numbers of clients connect to the server at once. Demonstrate that the server continues to work once the initial set of clients have completed and closed their connections to the server, and new connections are opened by later clients.