Grid Computing: Exercise 1

Dr. Colin Perkins

13 January 2005

The first week of the module has reviewed some concepts in networking and distributed systems, and introduced the problems of Grid Computing. This programming exercise seeks to review your understanding of basic network programming using the Berkeley sockets interface and the C programming language, as a precursor to the more advanced exercises later in the module.

You are to write a client-server application, providing a simple networked computation service. This service will use a simple but extensible textual encoding for operators and parameters, mimicing the structure that might be used in a simple RPC implementation. There are two parts to this application:

1. Write a server program which listens for a TCP connection, and receives commands which are formatted as "(f (int x) (int y))" (without the quotes) over that connection, where x and y are integers and f is a function to execute with those integers. The function f may be either + to signify addition or * to signify multiplication. On receiving such a command, the server will return the answer "(int z)" (where z is either the sum or the product of x and y) back to the client over the same TCP connection. For example, if sent "(+ (int 6) (int 2))" the server will respond with "(int 8)"; if sent "(* (int 3) (int 2)" the server will respond with (int 6)". If the command is malformed, or if any other command is received, the server should return the response "(error)". After processing a command the server should continue listening for new connections, allowing several clients to connect in turn (the server does not need to support multiple simultaneous client connections).

2. Write a client program which tests the operation of the server. This program should make several sequential TCP connections to the server, demonstrating that the server correctly performs addition using a range of inputs, and that the server correctly handles malformed input.

Both client and server must be written in the C programming language, using the Berkeley sockets API, and should run on Linux. You should submit printed source code for client and server, any makefiles or other build scripts you use, and a brief written design rationale.

Answers must be submitted by 5pm on 20th January 2006. A drop box will be available for submissions in Lilybank Gardens and submissions will only be accepted via that drop box. Ensure your name and matriculation number are included on a cover page attached to each submission, and attach a pink statement of originality form. This exercise will be marked either 0 or 1. A mark of 1 will be given for submissions that include code demonstrating an understanding of the Berkeley sockets API and C programming. A mark of 0 will be given if the submission does not demonstrate that a reasonable attempt has been made to gain such understanding (for example, if the code is seriously incomplete and clearly could not work).

Knowledge of C and socket programming is a prerequisite for this module. Those who are less familiar with the language should review any standard C programming text for background (I recommend B. W. Kernighan and D. M. Ritchie, "The C Programming Language", 2nd Edition, Prentice Hall, 1988, ISBN 0131103628). An introduction to the Berkeley sockets API can be found in W. R. Stevens, B. Fenner and A. M. Rudoff, "UNIX Network Programming (Volume 1: The Sockets Networking API)", 3rd Edition, Addison-Wesley, 2003, ISBN 0131411551. Alternatively, there are many tutorials available online.