TUM-2097 Project 1: Socket Programming

1 General Information

The socket programming exercise is designed to be done in teams of two students. You are allowed to work on it individually if you so choose. Feel free to use any code from the lecture without attribution. If you use code from any other source, you must add a comment clearly detailing what code you copied and from where.

This exercise consists of three parts. The first two can be implemented independently but should likely be tested together. For part three you must have the first two parts completed. The first two parts will each represent 50% of the grade for this assignement and the third part can be used to earn a 20% bonus.

Your programs will be graded based on correctness (80%) and style (coding style, documentation, build system). You must submit your solutions using Subversion by **January 1st 2010**. Late submissions will receive a score of zero unless you notify the instructor about a valid reason (such as sickness) **prior** to the deadline.

1.1 Subversion Access

In order to receive Subversion access, you must submit a request at

`https://projects.net.in.tum.de/projects/tum2097/register`.

In this request you must include the login name and desired password. Your login name should be your lastname followed by the last two digits of your student identification number.

Furthermore, each team should send an e-mail to `korsten@net.in.tum.de` to ask for a group directory. You should include a desired team name and the login names of the team members in the request. You are free to create teams of up to two students.

Once your account(s) have been created, you can access the repository at

`https://projects.net.in.tum.de/svn-ext/tum2097/TEAMNAME/`.

You must submit a request for account and team creation by **November 15th 2009**. We want you to use Subversion for development, not just submission! Note that only the version of your code that is HEAD in the repository on January 1st 2010 at 23:59 will be graded.

1.2 Error Handling

Your program must handle all possible errors, especially those from the various system calls. In general, an error (other than blocking) in the communication should result in the server closing the respective TCP stream. Furthermore, all error should be reported to `stderr` using appropriate log messages.

1.3 Coding Style

You should read the GNU coding standards for a reasonable guideline for writing good C code. In general, a shorter and simpler implementation will receive higher marks than a complex and long implementation. Naturally, comments and testcases will not be counted against you when we evalute the size of the code (in fact, good testcases and comments may earn you style points).

Your implementation should avoid (unnecessary) copying of data, use the minimum number of system calls, avoid busy-waiting and finally support a large number of concurrent connections. In general, all of these goals can be achieved using reasonable good coding style.

2 A TCP Server

For the first part, you are to implement an IPv4 TCP server in C/C++ that forwards all data on incomming connections to a stream-oriented UNIX domain server, and forwards replies from the UNIX domain server to the TCP server. Your program should take two arguments. The first argument is the port number on which the TCP server should listen for inbound connections. The second argument should be the path name for the UNIX domain server. You are free to choose your method for implementing the forwarding mechanism (example methods discussed in the lecture include threads or `select`).

Note that if your server receives a signal that one end of the TCP connection has been closed for reading or for writing (but not both), it should continue to forward data in the other direction and also signal the other end of the connection the partial closure using the appropriate `shutdown` call.

Your server should terminate only after receiving a typical process termination signal (i.e. `SIGTERM`, `SIGHUP`, `SIGKILL`) but not on pipe errors (`SIGPIPE`).

3 A UNIX Domain Server

For the second part, you are to implement a stream-oriented UNIX DOMAIN server in a language of your choice that forwards all data on incomming connections to an IPv4 TCP server. Your program should take two arguments. The first argument is the path name for the UNIX domain server. The second argument should be the hostname and port number to which connections are forwarded.

4 IPv6 extension

For the third part, you are to extend both of your server processes with full support for dual-stack IPv6.