Jonah Brooks
CS311 HW4
03-16-2012

1 Finding Perfect Numbers

In this project, I used sockets to communicate between three different programs: one server (manage.py), one report program (report.py), and one compute program (compute.cpp). Between these three functions I am able to calculate perfect numbers up to 2^{32}.

2 Design Decisions

One of the major design decisions I made for this project was in how to transmit data over sockets. I considered multiple approaches and finally settled on encoding data as bytes using bitwise operators before sending, and decoding the same way after receiving. This let me transmit data using only 4 bytes per message. I used certain sentinel values to identify new clients, then sorted them into lists. I used select to multiplex the incoming messages, then used the lists to determine which client I'm handling. From there, I used sentinels again in order to specify which request the client has sent. The report process has sentinels for requesting a report, and sentinels for issuing a kill command to manage. The compute process has sentinels for requesting a new range to check, and for telling manage that it has finished everything and does not wish to receive a new range. Any other value is interpreted as a verified perfect number.

Another major decision I made was in how I generated and sent reports. I chose to use the struct module to pack and unpack each perfect number with its corrosponding finder. This made it much easier to send a fixed amount of bytes over the socket without having to custom format my own code, and without having to figure out whether I had sent a number or a string. The struct module made this process much easier.

Also I decided to only let the user specify the hostname from command line, rather than host name and port. This is because I am only accepting connections from a single port on my server (to prevent conflicts with other users of os-classs), so there is no need to specify a different port number when running compute or report.

Lastly, I was unsure of how to get my server's external ip address, so I settled on making a temporary socket to connect to google.com over the http port, then use getsockname() to find the ip address over which I have connected. I'm sure there must be a more elegant way to do this, but it works just fine.

3 Issues Encountered

I ran into multiple issues while working on this project. Some required a small amount of work to overcome, such as issues with encoding and decoding my messages, while others proved more difficult than expect. Ultimately the largest issue I ran into was in creating a thread to wait for a kill signal from the server and terminate the compute process. My original plan was to create a new socket which communicates with the server, then wait on a recv until it hears anything at all, or the server issues a shutdown, then signal kill. However, I was not able to create another socket connecting to the same host on the same port, which, in hindsight, makes quite a bit of sense... However, I did not end up with enough time to implement a new solution. I'm still not really sure how to do this without a large restructuring of how I receive messages in my compute code.

However, in attempting to overcome these issues, I learned a great deal about sockets and other python and C++ features, so I am glad I spent the time working on them even without perfect results.

4 Commit Log

Commit Time	Commit Message
Fri, 16 Mar 2012 20:12:21 -0700	All done. It only seems to work from computer within the network that is running the server, but oh well. Also, I wasn't able to get the manage.py -¿ compute kill signal to work correctly. I have commented out the code that was attempting to implement that, and will mention it in the writeup.
Fri, 16 Mar 2012 00:02:01 -0700	Everything is finished except for the ability to have report kill manage, and manage kill compute. Everything else is working as I want it to.
Thu, 15 Mar 2012 07:50:16 -0700	Forgot to commit last night. I got both sides encoding and decoding messages properly, and a range request/reply function set up. I can now run the program completely using one client.
Tue, 13 Mar 2012 22:54:30 -0700	Added functionality to compute.cpp. It can now tell how many operations it can perform in a second, and it can encode and decode unsigned ints into character arrays that can be sent via sockets. I also refactored the code to make it more readable.
Mon, 12 Mar 2012 23:02:43 -0700	Initial commit. Basic parrot function between python server and c++ client. IOPS calculations and perfect number checker implemented in c++.