CS311 Project 4: Sockets and Python

Ryley Herrington

March 16, 2012

1 Introduction

This project required me to find perfect numbers and pass them to a server hosted on a different machine. I also had to implement a report client and a manage server that would be written in Python. The manage server had to be able to open a port on a machine and talk to a bunch of clients non-dependent on their language. Report has to be able to tell us what values we've already computed, who's computing right now, and it has to be able to kill the server.

2 Manage.py

2.1 Overview

This is the controlling process for a multi-process "perfect number" calculator. It is composed of 3 parts: manage - this "server" process report - the user interface client compute - the computation engine client

All socket communication between the processes is composed of single letter commands with character string arguments (in order to avoid any endian/byte-swapping issues).

2.2 Description of Operation

The overall data flow is that "manage" will start first and begin accepting commands.

Each "compute" client will register itself with "manage" via an "I" command, providing its own machine name, current PID (to differentiate multiple "compute" clients on one machine), and a measure of its operations per second. "manage" will respond with the first range of numbers to test.

As each range is completed, "compute" will contact "manage" with a "P" command to report any perfect numbers it has found, and "manage" will respond with a new range to test.

When any "report" process starts, it will make a request to "manage" and receive a single response.

To support the KILL command, the "manage" server will save the socket to each compute client in a dictionary named K_Socket. The client will be expected to monitor this socket for as long as it is executing.

Note that in all cases, the protocol is composed of: - a single command from a client to a server - a single response from server to client

Table 1: Description of Socket Protocol

Client. #	Command	Server	Response
report	N	manage	list-of-perfect-numbers
report	R	manage	count-of-numbers-tested
report	C	manage	list-of-compute-clients
report	T	manage	T [echo test]
report	K	manage	K [sends K to all compute clients]
calculate	I system pid OPS	manage	range to compute
calculate	P list of perfect numbers	manage	range to compute

3 Report.py

3.1 Usage

report.py with options:

S connect to 'manage' server on host arg
N report current list of perfect numbers
R report count of numbers searched so far
C report compute clients in use
K kill manage server and all compute clients

3.2 Explanation

This file is easy to understand because of the brevity of the file. Set up the interrupt handlers to deal with the signals defined in the project. Then I made the command line args to deal with s, T, N, K, R, and C. 's' will sleep the report.py client, only useful when I run a test. The T is good for testing to see if I can talk to other clients because it just prints out whatever it receives. The N will report how many numbers found so far. The K will kill everything. The R will tell how many numbers we've sent, and the C will tell who's running and where. Finally we kill the socket. Pretty simple.

4 Compute.c

4.1 Compute IOPS

compute_OPS : compute operations per second for this machine

We could measure the system we're on using IOPS or FLOPS, but it seems more practical to measure it based on the operation that we'll be performing. Thus, our measured "operation" will be a "value checked as a factor," which is probably dominated by the time for a mod operation.

Since we're insisting on the 'brute force" approach, when testing if N is perfect, we at least conceptually test each of the (approximately) N values below it to see if they are a factor. It doesn't matter that we might explicitly only test half of them, we're still determining if any of the (approximately) N values less than N are factors.

Since it takes N operations to determine if N is perfect, then it will take N*(N+1)/2 operations to search all of the numbers up to and including N. To simplify things, let's approximate that as N$\hat{2}$/2.

This means that to search a range from L to H, we would perform approximately H$\hat{2}$/2 - L$\hat{2}$/2 operations. If we perform this calculation and measure it, we can divide by the time to produce an "operations per second" metric.

The server can then use this metric to determine the appropriate size of range to pass to this calculation client (in order to keep the calculation in the 15 second range).

Given: OPS = operations/second (provided by "compute" client) L = low bound (lowest unsearch value)

Find: H = high bound of range to give to "compute" client

Solve for "H": (H$\hat{2}$ - L$\hat{2}$) / 2 = (15 * OPS) H$\hat{2}$ - L$\hat{2}$ = 2 (15 * OPS) H$\hat{2}$ = L$\hat{2}$ + 2 * (15 * OPS) H = sqrt(L$\hat{2}$ + 2 * (15 * OPS))

So, when the server is assigning a new range, it should use the smallest unassigned value for the low bound and use the previous formula to determine the upper bound.

4.2 Creating I Command

construct_I_command : Create the initialization command for registering a new compute client
This command will be composed of:
– The command character "I"
– The name of the system running the compute client
– The process id of the compute client
– The value of the computed operations per second
For example, it might look like: I mymachine 12345 1000000.0
Because all data is in string form, byte ordering is not an issue.

4.3 Creating P Command

construct_P_command : Create the command to report computed perfect numbers
This command will be composed of:
– The command character "P"
– Each of the perfect numbers found within the given range
For example, it might look like: P 6 28 496

4.4 Main

Basically in main I set up the signal handlers and their respective receivers. Then I create a deadman monitor to make sure I don't screw up and forget to kill a process. So if I run for more than 6 hours without receiving a command then I shut down automatically.

Then I register the compute client with an "I" command because I need to find out how many iops I can do per second. I figure it out with the method above with a certain range. Then while I have a number still to compute, I will listen to see if I get a kill command and then send how many IOPS I can do. From then on I will just receive a range and compute perfect numbers.

4.5 Perfect Numbers Calculation

Within the range passed into this method, we will check to see if it's divisible by any numbers. Then we add those to a candidate list. If the list adds up to the number we're checking then it's a perfect number and we add it to the "Perfect numbers list." Pretty simple, albeit slow and the worst way to do it.

5 Revision Control Log

Table 2: Revision Control Log

Rev. #	Date	Description
1.8	3/14	Final revision. Everything seems to work, even through firewalls.
1.7	3/13	Implemented all of Report. It uses all the right signals.
1.6	3/12	Manage now deals with kill command properly and sends back 13-18 seconds range perfectly
1.5	3/9	Added thread to compute to watch for kill.
1.4	3/9	Report added and seems to work.
1.3	3/8	Added perfect numbers and IOPS... it seems to work... check though
1.2	3/5	Added compute, and it talks to manage!
1.1	3/5	Initial revision – Python set up server – not sure if works yet.