Project 6: Sockets, python, and the client-server model

Due: Monday August 31, 2014 11:55 PM Pacific USA time zone.
Points on this assignment: 100 points.

Work submitted late will be penalized as described in the course syllabus. You must submit your work twice for this and all other homework assignments in this class. Ecampus wants to archive your work through Blackboard and EECS needs you to submit through TEACH to be graded. If you do not submit your assignment through TEACH, it cannot be graded (and you will be disappointed with your grade). Make sure you submit your work through TEACH and Blackboard. Submit your work for this assignment as a single tar.bzip file through TEACH and Blackboard. The same single tar.bzip file should also be submitted through Blackboard.

In all your source files (.c files, .py files, .h files, .txt files, and even the Makefile), you need to have 4 things at the top of every file as comments:

1. Your name
2. Your OSU email address (ONID or engineering)
3. The class name and section (this is CS344-400)
4. The assignment number (this is homework #6).

If you omit the 4 header lines, you can expect to lose 20% on your grade. It is easy to do, so just do it.

Description

Place all of the files you produce for this assignment in a single directory called Homework6.

This assignment is not a significant step up in challenge and complexity from previous homework assignments. Still, don't expect to start this on the Sunday or even the Saturday before it is due and complete it on time. You'll be really REALLY crabby.

Remember that the programming work in this class is intended to be individual work, not group work.

Submission

When you are ready to submit your files for this assignment, make sure you submit a single bzip file. Review homework #1, problem #1 if you need a refresher on how to do this. If your file is not a single bzip file, you cannot receive points on this homework assignment.

Perfect numbers

This project will require that you compute perfect numbers. These perfect numbers are numbers such that the sum of their proper divisors is equal to the number itself. For instance, the proper divisors of 6 are 1, 2, and 3, which sum to 6.

You will write 3 related programs to manage, report, and compute results. You are required to brute force the algorithm. Yes, this is ridiculous, but the algorithm is NOT the point of this assignment.

compute's job is to compute perfect numbers. It tests all numbers beginning from its starting point, subject to the constraints below. There may be more than one copy of compute running simultaneously.

manage's job is to maintain the results of compute. It also keeps track of the active compute processes, so that it can signal them to terminate, including the host name and performance characteristics.

report's job is to report on the perfect numbers found, the number tested, and the processes currently computing. If invoked with the "-k" switch, it also is used to inform the manage process to shut down computation. report will query the server for information, provide that to the caller, then shut down.

Use sockets for communication, with manage as the "master" process, or in other words, the server.

In order for the compute process to know what values to check, it will query the server for a range of values which it will check for perfect numbers. The server should send back a range that will take approximately 15 seconds for the client to check. In order for the server to know this, the clients must send some sort of information about their performance characteristics. To determine these performance characteristics (think FLOPS/IOPS), you will make use of a timing loop, where you execute a known number of operations, and time how long they take.

All processes should terminate cleanly on INTR, QUIT, and HANGUP signals. When the -k is flag is used on report, report sends an INTR to manage to force the same shutdown procedure. You will have to implement a standard to use via sockets to tell the other side to send itself the signal, via kill.

You will be implementing manage and report using python, while compute will be done in C. compute must utilize a separate thread or process for monitoring the control socket for a shut down message from the server.

You are required to use XML for data exchange. Further, you are not allowed to use any libraries to construct or parse the XML. You must implement this BY HAND. This is an exercise in string manipulation, as I've noticed a lack in such.

You must use the following compiler flags for gcc as part of your CFLAGS in your Makefile and the CFLAGS must be part of your gcc command line.

```
-Wall
-Wshadow
-Wunreachable-code
-Wredundant-decls
-Wmissing-declarations
-Wold-style-definition
-Wmissing-prototypes
-Wdeclaration-after-statement
-std=c99
```

Notes and suggestions

1. Remember: plan the work, work the plan.
2. I will use the following command line to build your code. You know how to create `clean` target in your Makefile.
   `make clean; make`

2.1 Your code will probably initially generate a LOT of messages from the compiler using those flags. Go into your code and clean them up. You may want to read about the C extern keyword. And, extern isn't just for variables.

2.2 Your code should have a clean compile when using those command line options to gcc. Clean means no error or warning messages.

3. Grade breakdown:

Task	Points
Using "-Wall -Wshadow -Wunreachable-code -Wredundant-decls -Wmissing-declarations -Woldstyle-definition -Wmissing-prototypes -Wdeclaration-after-statement -std=c99" in the Makefile without warnings from the compiler (if warnings, then 0). A single Makefile that compiles both portions of the assignment. Have a target in your Makefile for clean and use it before you tar up the directory.	20
The working application.	80
Total	100 points

Things to turn in

Things to include with the assignment (in a single bzipped file):

1. C source code (.c and .h files) for the solution to the posed problems (all files).
2. A Makefile to build your C code.
3. Python source code (all files)

Please combine all of the above files into a single bzip file prior to submission.