Computer Networks - CSE 30264 - Project 2

Total Points	**125 points**
Goal	**Program a TCP client, program a multi-threaded TCP server**
Assigned	**September 30, 2014**
Due	**October 30 2014, by class time**
Grouping	**To be completed as a small group project.**

Overview

In this project, you will implement the client and server sides of a simple real-time sensor application. The sensors that you will read are USB-connected thermal sensors.

Your tasks are as follows:

- Write a client application which is run by cron. Your code will read a configuration file, read the sensors, and transfer the readings to your multi-threaded server. See the client-side design section for more information on the client code.
- Write a multi-threaded TCP server which collects the readings from the sensors, and saves the readings in the appropriate data files. See the server-side design section for more information on the server code.

Structure

Your code should be comprised of multiple source files. Shared variable definitions should be contained in ".h" files which are included by your client and server code. You need to include Makefiles or autoconf files which can be used to create the appropriate Makefiles and binaries.

Client-side Design

1) The driver code in the ***/afs/nd.edu/coursefa.14/cse/cse30264.01/files/Project2/newgo.c*** file may be compiled and run as it stands. This code will read the sensors and spit out two floating point numbers. I recommend that you copy pertinent portions of the driver code into your program (giving attribution to the author(s) of the original code), and modify the code as required in order to accomplish the requirements detailed in the client-side design specifications, as well as to **ensure that the code is multi-thread safe**. There is a binary of the newgo code in the same directory as the source code. The binary, **newgo_bin** will read the sensors, and print the temperature information to the screen.
2) The sensors on the student machines are named **"/dev/gotemp"** and **"/dev/gotemp2"**. Your code should handle the case of zero, one, or two sensors on a host.
3) Each of the student machines contains a client configuration file. Your code should read the configuration file to determine how many sensors exist on this host, and acceptable temperature ranges for each sensor. The configuration file is **/etc/t_client/client.conf**.
4) The format of the **client.conf** file is:
   Line 1: int: number of sensors
   Line 2: double{space}double low and high values for sensor 0
   Line 3 through Line N: double{space}double low and high values for sensors 2 through N.
5) Your client code should read the thermometers every **30** minutes under the control of the **cron** service. Because the **supplied** sensor code is **not** thread safe, you need to make sure that your code is thread safe, and that (when run under cron) only reads the thermometers at the times given in Appendix A.
6) You need to create a data structure to hold the following information (in this order):
   - **Name of the host with the sensors (32 character string)**
   - **number of sensors attached to this host (int)**
     - read **/etc/t_client/client.conf** for this information.

- **sensor number (int),** /dev/gotemp == sensor 0, /dev/gotemp2 == sensor 1
- **sensor data for this sensor (double) ,**
- **acceptable low value for this sensor (double) (from config file)**
- **acceptable high value for this sensor (double) (from config file)**
- **timestamp (32 character string)** You can obtain a timestamp using the system **timestamp** function.
- **action requested (int)**
  - For now there are two actions: Send sensor data, (value = 0), and request status (value = 1). Sensor data fields in your struct should be set to zero for a "request status" packet.
  - **NOTE: Each packet will contain data for one sensor. If the host has two sensors, you will need to send two packets to the server**

7) Your code needs to fill in your data structure, connect to the server (setting the action field to zero to signify that you are sending readings to the server), and transfer the readings to the server.
8) If there is an error condition (cannot open sensor, ...) your code should write a self explanatory message to **/var/log/therm/error/group#_error_log** (for example g0_error_log).
9) Your client should then exit until awakened by cron.
10) Name your client binary "**therm**"

Server Side Design

1) The server is responsible for handling the connection request from clients, processing the request, and looping back around to handle future requests.
2) Your **multi-threaded** server should bind to a TCP port between 9000, and 10000 (See Appendix A for port assignments) and listen for incoming client connections. You may decide if you would like to allow timeouts for better responsiveness but any sort of a timeout is purely optional. You may want to allow for port reuse for quicker recovery after a crash.
3) Once a new client connection arrives, use the accept to create a new client socket as well as a new thread.
4) Read the data from the client, and parse it.

   If the action == zero, this is a new reading which need to be stored in the appropriate file.
5) If the packet contains new readings, you need to append the data to a logfile in **/var/log/therm/temp_logs/** as follows:

   The file name is your group# followed by an underscore followed by the year followed by an underscore, followed by the numeric month, followed by an underscore, followed by the name of the host i.e.:
   /g00_2014_09_student00

   The format of the logfile should be as follows:
   Year{space}month{space}day{space}hour{space}minute{space}sensor0_reading{space}sensor1_reading
   for example: **2014 09 30 01 41 67.66 94.37**
6) Name your server binary **thermd.**

Testing

There are four student machines. Once you think you have your code working, you should start your cron-driven client on all four student machines. This will allow you to test the multi-threading capabilities of your server, as all four machines will attempt to send updates to the server at the same time.

Submission

Submit a gzipped tar file (**proj2.tar.gz**) of your entire project package to the Project2 directory of **one** of the

group members. You tar file should create two separate directories under the dropbox/Project2 directory, a **server** directory for the server code and a **client** directory for the client code. The names of all team members **must** be included as a comment in **each** source file.

Each directory should contain a Makefile for building your code appropriately, be that through g++ or gcc.

The tar archive must include the following:

- **Working client code in a client subdirectory**
  - Makefile for the client
  - Source file for the client
  - Appropriate comments with team member names
- **Working server code in a server subdirectory**
  - Makefile for the server
  - Source file for the server
  - Appropriate comments with team member names
- **README file**
  - Team member names
  - Listing of the included file names in the archive
  - Design decisions and problems encountered/solved

Grading Rubric

- Client code (45 points)
  - [5 pts] Included Makefile compiles code without errors.
  - [5 pts] Code follows argument convention, and reads configuration file.
  - [10 pts] client code which reads sensors is thread safe
  - [10 pts] Code transfers the readings to server.
  - [5 pts] Code provides reasonable error checking.
  - [5 pts] Reasonable code performance.
  - [5 pts] Code is commented with appropriate header information.
- Server code (45 points)
  - [5 pts] Included Makefile compiles code without errors.
  - [5 pts] Code follows argument convention.
  - [5 pts] Code reads sensor data from client.
  - [10 pts] Code saves sensor data in appropriate files.
  - [5 pts] Code provides reasonable error checking.
  - [5 pts] Reasonable code performance, no memory leaks.
  - [5 pts] Code is commented with appropriate header information.
  - [5 pts] README file which describes your code, design strategy, and other pertinent information.
- Overall (25 points)
  - [10 pts] Code is submitted in requested form
  - [15 pts] README file which describes your code, design strategy, usage information, and crontab used to invoke the client code.

Extra Credit

If your code all works, and you would like to earn extra credit, you can add in the following functionality. You will only receive credit for one of these options, and then **only if your client/server code operate properly:**

- [25 pts] If the readings create an overtemp condition (sensor reading is greater than the “high” value from the client config file), send a status message to the client. This will require code in your client to read the status, as well as code on your server to check the reading against the high-value, and to send a status message if there is a problem. The client can request a status packet by sending a packet to the server with the action field set to 1.
- [25 pts] Use ***rrdtool*** to create an annotated graph of the sensor data stored in the files located in the **logs** directory.

If you elect to do the extra credit, include a note appropriately in your README.

Appendix A

Note: The sensor driver code I supplied is not thread safe. If multiple reads occur simultaneously, you may get errors from the sensor (can't open, no such device, odd temperature readings, ...). Restrict your cron driven sensor reads to the times given in Table 1. In order to avoid port conflicts, I assigned TCP port numbers to each group. Your client and server should communicate using this port.

Table 1. Group Port/Read Time assignments for cron jobs

Group	TCP Port	Read time	Notes
g01	9762	0,30	
g02	9763	1,31	
g03	9764	2,32	
g04	9765	3,33	
g05	9766	4,34	
g06	9767	5,35	
g07	9768	6,36	
g08	9769	7,37	
g09	9770	8,38	
g10	9771	9,39	
g11	9772	10,40	
g12	9773	11,41	
g13	9774	12,42	
g14	9775	13,43	
g15	9776	14,44	
g16	9777	15,45	
g17	9778	16,46	
g18	9779	17,47	
g19	9780	18,48	
g20	9781	19,49	
g21	9782	20.50	
g22	9783	21,51	
g23	9784	22,52	