CMPSC 311 Assignment 5: Device Driver Network Support

Out: April 11, 2014 Due: May 2, 2014

Purpose

This assignment is designed to give you some basic experience in socket programming by implementing an application-layer protocol. You will take an existing device driver and add network support by sending requests to a server which controls the device.

Note: a superficial reading of this assignment may lead you to believe that the work is easy, but it is not. Please allow ample time to complete the assignment. Since it is due the last day of class, no late submissions can be accepted, and no extensions can be granted.

Description

In this assignment you will extend the MultiTron driver to communicate over the network with the MultiTron, sending and receiving commands and data. The modified driver will use a client/server networking model, in which all requests must be sent over the network to a server and responses are sent back to the client. In this way, the driver could support controlling a MultiTron located anywhere on the Internet from your host computer.

The protocol used to communicate between a MultiTron client and server consists of two messages. The *MultiTron request message* is sent from your client program to the server: it contains an instruction, and a data buffer when needed. The *MultiTron response message* is sent from the server back to your client: it contains a return value, and a data buffer when needed. Both messages use the same format (offset and size are in bytes):

Offset	Size	Description
0	4	MultiTron instruction (the `op` parameter from `tronctl`)
4	2	Number of bytes n in the buffer portion of the message
6	2	Return value
8	n	Data buffer

Figure 1: MultiTron message format

All multi-byte values in a MultiTron network message are in *network byte order* (big-endian). You must use appropriate functions to convert the values, as doing the conversion manually is generally not portable to other systems.

Programmatically, all calls to `tronctl` have been replaced with calls to `tronctl_net`. This function should send the appropriate message to the MultiTron server, then receive and handle the response. The `tronctl_net` function will be written by you. This function is called as follows:

```
int tronctl_net(int *sock, uint32_t instr, void *data, int datalen);
```

where the parameters are:

- `sock` (passed by reference): File descriptor of a socket which is connected to the server.
- `instr`: MultiTron instruction (the parameter formerly known as `op`). The format of this parameter is exactly the same as in previous assignments.
- `data`: Data buffer for the operation, if needed. This parameter should function exactly as in previous assignments.
- `datalen`: Size, in bytes, of the data buffer.

If the `sock` parameter is an invalid file descriptor (negative), then the socket is not yet connected, and the function should first connect to the address and port supplied by the constants `MULTITRON_DEFAULT_IP` and `MULTITRON_DEFAULT_PORT`, storing the corresponding socket descriptor in this parameter for later use. Once connected, the function should construct and send a request message to the server, as described above, and wait for the response. The response will have the return value and an optional block of data. The opcodes, instruction format, and data buffer are exactly the same as we have seen in previous assignments.

The return value from `tronctl_net` should be zero if the function executes successfully, and nonzero if there was an error (including one indicated by the return value from the server).

Simulation and verifier

The `client` and `server` programs included in the source tarball will run a simulation of a networked MultiTron. Both programs can be run with the `-h` option to print a usage message. The `server` program handles requests from clients until it receives a SIGINT (Ctrl-C), SIGHUP, or SIGTERM signal. The `client` program reads a workload from standard input, just like the `simulate` program from previous assignments, and executes the corresponding driver commands using your `tronctl_net` implementation.

The provided `verify` script will run a number of unit tests on the implementation of the network driver, using the workloads found in the `workload` directory. For each test, it will print whether the test has succeeded or failed. The verifier will also test for correct handling of error conditions.

By default, the `verify` script will stop after the first error, leaving the log files `mtclient.log` and `mtserver.log` in the current directory for your inspection. If you would like it to continue to attempt all of the tests, give the `-a` option to the script.

If you cannot determine the source of the problem from the output of the verifier and inspecting any relevant log files, you will need to use a debugger to locate the issue.

Procedure

1. Log into your virtual machine and open a terminal emulator.
2. Download the starter code and extract the tarball as you did in previous assignments:

   `http://www.cse.psu.edu/~djp284/cmpsc311-s14/docs/assign5.tar.gz`

   This will create a directory `assign5` containing these files:

   - `nettron.c`: Network implementation. All of your code goes here. *This is the only file you need to edit.*
   - `nettron.h`: Header file containing function prototypes and constants for the code you will be writing. You should not change this file.

- `libmtnet.a`: MultiTron and driver implementation, provided as a static library. (This is the 64-bit version; on a 32-bit machine, edit the Makefile to include `libmtnet32.a` instead.)
- `client.c` and `server.c`: Stubs which run the client and server processes.
- `Makefile`: Makefile for the entire project. You can edit this if needed.
- `verify`: Script which runs client/server unit tests.
- `workload/`: Directory of unit test workloads.

3. Implement the `tronctl_net` function in `nettron.c`, commenting all of your code as you write it to explain how it works. You may create auxiliary functions to make your code cleaner; if you do so, please place them in `nettron.c` as well.

Submission

1. Run `make clean` to remove any build products (binaries, object files, etc.) from the directory.
2. Create a tarball containing the `assign5` directory, complete with the source code you have written. The tarball should be named `LASTNAME-psuid-assign5.tar.gz`, where `LASTNAME` is your last name in all capital letters and `psuid` is your PSU email ID (the one you would use to log into ANGEL). For example, if the instructor were submitting this assignment, he would run:

```
$ tar -cvzf POHLY-djp284-assign5.tar.gz assign5/
```

3. Verify that the tarball contains all of your source code files by viewing the file list:

```
$ tar -tvzf LASTNAME-psuid-assign5.tar.gz
```

4. Before sending the tarball, test it in a clean temporary directory using the following commands:

```
$ mkdir /tmp/assign5test
$ cp LASTNAME-psuid-assign5.tar.gz /tmp/assign5test
$ cd /tmp/assign5test
$ tar -xvzf LASTNAME-psuid-assign5.tar.gz
$ cd assign5
$ make
$ ./verify
```

5. Send the tarball in an email with the subject line "311 submission" to both the instructor (`djpohly@cse.psu.edu`) and the TA for your section (listed on the course website). Put your own email in the Cc: line so you can be sure the email was sent properly and contains everything. **YOU ARE RESPONSIBLE FOR DOING THIS CORRECTLY.** Lost emails, emails not sent to both the instructor and TA, bad tarballs, and missing files are the student's responsibility and will be treated as late or missing. There will be no exceptions.

   This email should be sent by 11:59pm of the due date of the assignment; late submissions will **NOT** be accepted on this assignment.

Reminder

As with all assignments in this class, you are prohibited from:

- Copying any content from the Internet.

- Discussing or sharing ideas, code, configuration, text, or anything else with others in the class.
- Seeking significant help from anyone inside or outside the class other than the instructor and TAs.

Likewise, you are responsible to protect your own code from falling into the hands of others. Consulting online sources is acceptable, but under no circumstances should *anything* be copied. Failure to abide by this requirement will result in dismissal from the class as described in the course syllabus.

Above all: when in doubt, be honest. If you do for any reason get significant help from a source, online, in person, etc., *document it in your submission* before we have to ask you about it. This will go a long way!