CMPSC 311 Assignment 5: Device Driver Network Support

Out: April 11, 2015 Due: May 1, 2015

Purpose

This assignment is designed to help you develop some experience in socket programming by implementing an application-layer protocol. You will take your existing device driver and add network support by sending requests to a server which controls the device. This server is provided, and you will be converting your driver into a network client which communicates with it.

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 your TipTap driver to send and receive instructions and data to a network-connected TipTap. The modified driver will use a client/server networking model, in which all instructions must be sent over the network to a server and responses are sent back to the client. After implementing this assignment, your driver will have the ability to control a TipTap located anywhere on the Internet.

The protocol used to communicate between a TipTap client and server consists of two messages. The *TipTap request message* is sent from your client program to the server: it contains a TipTap instruction as well as a data buffer if needed. The format of a TipTap request message is as follows (where offset and size are in bytes):

Offset	**Size**	**Description**
0	4	TipTap instruction (the `insn` parameter from `tapctl`)
4	2	Number of bytes n being sent in the data buffer
6	n	Data buffer

Figure 1: TipTap request message

When the server receives a request, it will read the instruction and data, then execute the corresponding `tapctl` request locally. The results of this request will be sent back to the client in a *TipTap response message*, which is formatted as follows:

Offset	**Size**	**Description**
0	2	Return value
2	2	Number of bytes n being sent in the data buffer
4	n	Data buffer

Figure 2: TipTap response message

The following fields in TipTap network messages are transmitted in *network byte order* (big-endian): instruction, number of bytes, and return value. Assume the contents of the data buffer can be transferred without modification. As in previous assignments, you must use appropriate functions to convert these values, because doing the conversion manually is not portable to different systems.

You will be implementing a new version of the `tapctl` function which uses the TipTap message protocol to communicate with a server that controls the TipTap. Each call to this function will construct a single request message, send it to the TipTap server, then receive and handle a single response. It will also be responsible for establishing and closing the connection to the server when needed. As a reminder, the function signature is:

```
int tapctl(uint32_t insn, void *data);
```

where the parameters, opcodes, flags, instruction format, and data buffer are all the same as in previous assignments.

This networked `tapctl` function should behave as follows:

- If the opcode of the `insn` parameter is `TT_POWERON`, then the function should first connect to the address and port supplied by the constants `TIPTAP_IP` and `TIPTAP_PORT`. Once connected, the file descriptor for the socket should be stored in the provided global variable `sock`.
- The client should construct a request message using the provided `insn` parameter and, if needed, the data in the `buf` parameter. This message is sent over the socket to the server.
- The client will then receive a response message from the server. The return value field indicates whether the TipTap command succeeded, and any data that should be returned to the caller will be sent back in this message.
- Finally, if the opcode of the `insn` parameter is `TT_POWEROFF`, the function should disconnect from the server after receiving the response.

The return value from `tapctl` should be zero if the function executes successfully, and nonzero if there was an error. This includes errors in the network functions, as well as errors indicated by the return value from the server.

Simulation and verifier

The `simulate` and `server` programs included in the source tarball will run a simulation of a networked TipTap. 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 `simulate` program acts as a client which reads a workload from standard input, just like in Assignment 3, and it executes the corresponding driver commands using your `tapctl` implementation. When testing, you should start the server first and leave it open in one terminal window, then run the client in a second terminal. If one or both get stuck waiting for messages, use Ctrl-C to kill them.

The provided `a5verify` script will run the unit tests from Assignment 3 using the new networked driver, using the workloads found in the `test` 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. As with `simulate`, you need to have the server running already when you run `a5verify`. If you cannot determine the source of the problem from the output of the verifier, you will need to use Wireshark and/or GDB 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-s15/docs/assign5.tar.gz`

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

   - `network.c`: Network implementation. *This is the only provided file you need to edit.*
   - `network.h`: Header file containing function prototypes and constants for your code.
   - `a5verify`: Script which runs client/server unit tests.
   - `test/`: Directory of unit test workloads.
   - `libtiptap.a` and `libttnet.a`: TipTap hardware and server implementation, provided as static libraries. The Makefile will automatically detect if you need the 32-bit versions.
   - `simulate.c` and `server.c`: Stubs which run the client and server processes.
   - `tiptap.h`: Header file containing constants for TipTap opcodes, flags, etc.
   - `Makefile`: Makefile for the entire project.
3. Copy your `driver.c` and `driver.h` files into the `assign5` directory. You should be able to compile the project at this point.
4. Implement the new `tapctl` function in `network.c`, commenting all of your code as you write it to explain how it works. You may create auxiliary helper functions to make your code cleaner; if you do so, please place them in `network.c` as well.

Submission

1. Run `make clean` to remove any build products (binaries, object files, etc.) from the directory.
2. Run `make tarball` to create a tarball in the parent directory of `assign5`.
3. Rename this tarball to `LASTNAME-uid1234-assign5.tar.gz`, where `LASTNAME` is your last name in all capital letters and `uid1234` 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 call the tarball `POHLY-djp284-assign5.tar.gz`.
4. Before sending the tarball, test it in a clean temporary directory using the following commands:

```
$ mkdir /tmp/assign5test
$ cp LASTNAME-uid1234-assign5.tar.gz /tmp/assign5test
$ cd /tmp/assign5test
$ tar -xvzf LASTNAME-uid1234-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.

Important reminder

As with all assignments in this class, you are explicitly not permitted to:

- Copy any content from the Internet.
- Discuss or share ideas, code, configuration, text, or anything else related to the assignment with others in the class.
- Solicit help from *anyone* inside or outside the class other than the instructor, TAs, or assistants explicitly permitted by the instructor.
- Allow your own ideas and code to be seen or used by others. You are responsible for protecting your own work.

If you are unsure of what is permitted, you should always ask the instructor *before* doing it. If you need help on the assignment, you should contact one of the TAs.

You may be asked by the instructor or TA to explain portions of your code. If you cannot explain how or why your own submission works, it will be considered plagiarized. 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 with an F as described in the course syllabus.

Above all: when in doubt, be honest. If you do for any reason get significant help online, in person, etc., or use code from a source such as previous projects or other courses’ lecture notes, please *document it in your submission* before we have to ask you about it. This will go a long way!