Captain's Log

Author: Emilio Lopez, eil11

For EECS 325 Computer Networks I, with Professor Mark Allman

Captain's Log is a client-server application protocol that enables a captain of ship – be it space or pirate – to record personal notes onto a server, safely away from his/her crew.

As mentioned, the protocol is client-server. The captain's client sends commands to the captain's server, which processes the commands and relevant arguments. The protocol relies on plain text transmission, which means a captain must ensure that none of his crewmembers are monitoring his network traffic.

Usage:

1. Start up the server using this command: ./proj4d <PORT_NUMBER>
    a. The port number is one of the captain's choosing.
2. Start up the client using this command: ./proj4 <SERVER_HOST_NAME> <PORT_NUMBER>
    a. The <PORT_NUMBER> here must be the same as the <PORT_NUMBER> in Step 1.
3. The client will now await commands from the captain. They are all in the form: <SINGLE_DIGIT> <COMMAND_ARGS>, unless the command uses no arguments.
4. To quit the client, simply input a number NOT in the range of [1 – 6]. Recommended is 0 or 7 for simplicity of memory.

Transport Protocol:

Captain's Log relies on Transport Control Protocol (TCP) for many reasons:

1. Message integrity is **critical**. If some information is lost or if the sequence of the information is distorted, the log entry will make no sense to the captain, if it even arrives to the server. TCP relies packets containing sequence and acknowledgement numbers to make sure that packets containing the captain's message arrive in the proper order, and that lost packets are retransmitted.
2. Reliability even over slower networks. TCP has built in congestion control, which helps with determining a good upload/download speed for the packets being exchanged between the captain's client and the server. When at sea or in deep space, there is no guarantee of high speed internet.

TCP guarantees in-order and reliable delivery of the information, whereas UDP cannot as it relies on best effort to transmit information.

Commands:

All commands have a format of <NUMBER> <COMMAND_ARGS>, unless the command requires no arguments.

Note: If a command requires arguments, the arguments should start after the space. Although the command will still run if there is any kind of valid argument after the (intended) space, information will be distorted.

For example:

```
C-> 1THIS WORKS NOT WELL

C-> 2 0

S-> HIS WORKS NOT WELL
```

If a command requires that there are no arguments, an attempted usage of that command with arguments will fail, and prompt the user to try again.

If a command requires that there are arguments, an attempted usage of that command without arguments will fail, and prompt the user to try again.

WRITE_LOG:

Input: `1 <LOG ENTRY>`

Output： `NONE`

WRITE_LOG writes the server the log information that the captain wants to keep.

For simplicity, WRITE_LOG places the new log in the next available index.

If the arguments exceed the max length, the client will try to process the character after the limit as a command. If that is a number and there is an “argument”, the log will double write; however, the client still functions without errors. Not a bug, a feature.

For example:

> After 3 writes, the next index that will be written in is index 3.
>
> If the log is cleared and another write occurs, the log will be written to index 0 and the next index will be written in index 1.

When WRITE_LOG successfully writes, it updates the number of logs in the server and the most recently made log.

If after writing to the captain’s log, the max number of logs has been reached, the server will archive the log and clear all the entries from it.

READ_LOG:

Input: `2 <LOG INDEX>`

Output: `The contents of the log at that index`

Since the captain is expected to be computer science savvy, the input should follow the rules of indexing an array. That is to say, the first index is 0 and the last is 364.

If the captain enters an index to read that is empty, the server will write that the log index is empty.

CLEAR_LOG:

Input: `3`

Output: `"Log Clear command sent"`

This command completely deletes every log entry from the Captain's Log. This, obviously, resets the number of logs in the captain's log.

This is best used after archiving the log.

NUM_ENTRIES:

Input: `4`

Output: `The number of logs in the captain's log`

This command simply gets the number of logs in the captain's log. Since this is stored as static variable in the server, it does NOT have O(N) complexity.

LAST_MADE:

Input: `5`

Output: `The most recently made log`

The server sends to the client the most recently written log. This allows for ease of use by the captain to see what he/she just wrote.

ARCHIVE_LOG:

Input: `6`

Output: `Archive Log command sent`

When the captain believes that he/she has written enough logs to the server, he/she can write all the logs in the server to a text file that is the stored on the server.

The text file is written to the same directory where the server executable runs. It has the filename format of:

“`CaptainLogArchive\ YYYY-MM-DD\ HH\:MM\:SS.txt`”

Each log gets its own line.

QUIT:

Input: Any single digit number not in the range of [1-6] (inclusive)

Output: `quitting application`

This command ceases the connection between the client and the server.