Advanced Space - Software Assessment

Purpose

The goal of this assignment is to demonstrate that you can write code in C and Python. Adherence to the problem statement, quality of the code and its documentation will be evaluated. No need to future-proof the code or make it more complicated than needed.

Timeline

You may take as much, or as little time as you wish to demonstrate your skills and complete the assignment, however we suggest shooting for a few hours. We only ask that you deliver your solution within **4 days** of assignment (please reach out if that timeline doesn't work for you and we can figure something out).

Deliverables

We would like the following deliverables to be emailed to tyler.hanf@advancedspace.com as a compressed archive (zip, tar.gz, tar, etc.). Please also use that email address to request clarifications on this assignment

- A short report (single digit number of paragraphs) detailing design choices on serialization of the communication between the client and the server (cf. assignment below); any out of the ordinary design patterns used, if applicable; the estimated duration it took to complete this problem; improvements to the assignment itself; the test cases used to determine correct functionality.
- Documented source code needed to build and execute both the client and server.
- Building instructions (which may be in the report), with an optional Makefile or a similar ubiquitous build automation script.
- Instructions to operate the client for it to send commands to the server.

Please avoid compressing files related to the version control system you're using, if any, or any binary resulting from compilation.

Problem

Two tools must be developed: a state machine in C, and a tool to control it in Python.

These tools communicate over a network socket of your choice (TCP, UDP, etc.). It should **not** be assumed that both tools will be executed from the same computer. The request/response serialization between these two systems is also left up to the developer.

The state machine, herein FSW, must respond to seven commands listed below. All commands must be sent from the Python tool, herein GCS. This tool must also allow sending at least one invalid command to test for proper implementation of the requirements. The GCS must also print out the response from the FSW in a human readable fashion. All responses from the FSW must include the state of the FSW and the information returned by the command if applicable, as defined below. If an invalid command is sent to the FSW, it shall only respond with its state information.

There are four states in the state machine:

- Restarting
- Ready
- Safe mode
- BBQ mode.

On start, the FSW is in restarting mode. It then waits for ten seconds (time during which it is not expected to respond to commands), and switches to the ready state.

When in the ready state, it is expected to answer all commands as sent by the GCS. After 5 subsequent wrong commands sent by the GCS, the FSW shall switch to safe mode. After 3 more subsequent wrong commands, the system shall switch to BBQ mode.

Here are the available commands, how they should work, and when they should work (some commands are disabled when the FSW is in safe mode or BBQ mode, or both). The encoding of these commands into something interpretable by the FSW is left up to the developer.

- "**safe mode enable**" will switch the FSW to safe mode. This command is invalid when in BBQ mode or in safe mode.
- "**safe mode disable**" will switch the FSW back to the ready state. This command is always valid.
- "**show number of commands received**" will return the total number of valid commands received since startup of the FSW. This command is invalid when in BBQ mode or in safe mode.
- "**show number of safe modes**" will return the number of times the FSW has switched to safe mode since startup. This command is invalid when in BBQ mode.
- "**show up time**" will return the number of seconds since the FSW was started. Developer may use the system clock to determine this information. This command is invalid when in BBQ mode or in safe mode.

- "**reset command counter**" will reset the command counter to zero and then return the number of commands received (i.e. zero). This command is invalid when in BBQ mode or in safe mode.
- "**shutdown**" will return the state of the FSW, close the socket and stop the process. This command is always valid.