TVision Insights Device Team Coding Exercise

The following problem is intended to help us evaluate the way you design and implement solutions to technical problems. If you have questions about the problem statement, make an effort to choose assumptions that seem reasonable to you, and document them.

Please solve this problem using Python, assuming an Ubuntu Linux host. You may use open source libraries if they are reasonably common packages on PyPI. You should include any necessary instructions to allow us to run your code.

Please spend no more than four hours on this exercise, and let us know how long you spent.

Background

On the device we place in panelist households, we may have several data collection processes running at any given time. Sometimes we need to start, stop, or otherwise control one of these processes at a specific time of day, and have an audit log of the control actions that were taken. Your task is to write a simple scheduler capable of process control.

Problem statement

Your program should read a configuration file, whose content and format are documented below. It should then run until killed, taking actions to control and/or communicate with other programs as dictated by the configuration file. Every action taken by your program, successful or not, should be logged to stdout.

In the process of carrying out an action, your program *must not* block so long that it misses the next scheduled action. If this would occur (say, because it is trying to write to a socket with no reader at the other end), you should report failure of the action that blocked, and proceed with the remainder of the schedule. (You are also free to choose a reasonable timeout for failure that is *shorter* than the time remaining until the next scheduled action.)

The single command line argument to your program should be a path (absolute or relative is OK) to the configuration file: `myprogram config.json`

Configuration format

A configuration file consists of a list of directives, each of which contains a **time**, a **verb**, and some **additional data** which depends on the verb.

The time is always UTC (you may assume that the system clock is UTC), and is represented as a string in the format `HH:MM` (exactly five characters). The action specified by the directive is to be carried out *each day* at the time named in the directive. If for some reason it cannot be carried out, a log entry should be written to stdout. You may assume that at most one directive appears per distinct value of **time**.

You may write your program to accept its configuration in *any one* of the following formats: CSV, JSON, YAML. (That is, you don't have to support *all* of these formats; *choose* whichever one is most convenient for you, and write your program to support it.) Each configuration verb has its syntax described in each format.

Configuration directives

start

CSV
```
11:30,start,/srv/release-bunnies,/var/run/release-bunnies.pid
```
JSON
```
{"time": "11:30",
 "verb": "start",
 "program_name": "/srv/release-bunnies",
 "pidfile_name": "/var/run/release-bunnies.pid"}
```
YAML
```
time: "11:30"
verb: "start"
program_name: "/srv/release-bunnies"
pidfile_name: "/var/run/release-bunnies.pid"
```

At the time of day specified by `time`, check whether a file exists at absolute path given by `pidfile_name`. If it already exists and it contains the PID of a process which is still running, take no action other than logging. Otherwise, start the process named by `program_name`, and (overwriting if necessary) cause the file at path `pidfile_name` to contain the PID of the newly started process. Log any errors encountered to stdout.

stop

CSV
```
13:30,stop,/var/run/release-bunnies.pid
```
JSON
```
{"time": "13:30",
 "verb": "stop",
 "pidfile_name": "/var/run/release-bunnies.pid"}
```

YAML
```
time: "13:30"
verb: "stop"
pidfile_name: "/var/run/release-bunnies.pid"
```

At the time of day specified by `time`, check whether a file exists at absolute path given by `pidfile_name`. If it already exists and it contains the PID of a process which is still running, terminate this process and remove the PID file. Otherwise, take no action other than logging. Log any errors encountered to stdout.

write

CSV
```
12:30,write,/var/run/release-bunnies.sock,37
```
JSON
```
{"time": "12:30",
 "verb": "write",
 "socket_name": "/var/run/release-bunnies.sock",
 "message": "37"}
```
YAML
```
time: "12:30"
verb: "write"
socket_name: "/var/run/release-bunnies.sock"
message: "37"
```

At the time of day specified by `time`, attempt to open the file path specified by `socket_name` as a Unix domain socket, and write `message` to it (encoded as a UTF-8 string). You may use a synchronous blocking write as long as you do not miss the next scheduled event. Log any errors encountered to stdout.

Evaluation criteria

Strive for correctness and clarity, because code is read many times before it is ever run in production. In addition to inventing and coding solutions to problems, we want to see that you can read requirements, incorporate them into your work, and clearly communicate the results to your teammates.

Please submit your program as a `tar.gz` containing all relevant source code and documentation, by email to your contact for the position. If you are not able to email the tarball directly, you may also upload it to a cloud service such as Google Drive and send your contact a share link.

Good luck!