Author: Steven Vuong
Last Edited: 09-05-2020

Python Cron Job Implementation

Background:
Cron is a software utility which implements a time-based job scheduler. Fields are entered into a cron table which is stored as a file, and executes the command specified in the field/row when the time/date is matched. An example of a field might be:

```
@minute @hour @day_of_month @month @day_of_week <BASH_CMD_TO_EXECUTE>
```

Where:
- @minute (0-59)
- @hour (0-23)
- @day_of_month (1-31)
- @month (1-12)
- @day_of_week (0-6), Sunday is 0
- There is also @reboot, to run on start-up

Along with numerical inputs, one may input (*) to indicate non-specificity.

There are further nuances of which can be found at and of which we will explore in more detail below:
- https://en.wikipedia.org/wiki/Cron
- https://code.tutsplus.com/tutorials/scheduling-tasks-with-cron-jobs--net-8800

Overview:
We have implemented a cron-style service which runs on Python which can be run by passing command line arguments containing the bash command to execute. Basic requirements for Linux / MacOS is Python 3.6 or above. Other OS' have not been tested and neither has Python 2.7.
A template of running our implementation may be:

```
nice -n 19 python main.py -sc "<BASH COMMAND>" -ct "<CRON_TIME_SETTING>"
```

The `nice -n` may be omitted, and is a Unix capability which lowers the priority of the program for usage of CPU when demand is high https://en.wikipedia.org/wiki/Nice_%28Unix%29. This allows the program to be run without interfering with mainstream processes too much.

Another thing to note is that the bash command and cron time setting is enclosed in speech marks. An example is:

```
nice -n 19 python main.py -sc "echo hello world" -ct "* * * * *"
```

Which outputs "hello world" to the command line at the beginning of every minute.

System Design:
Our implementation consists of two primary components.

1. The first is the command line argument parser. This parses command options of:

   -r : reboot, Currently has no function. Original idea is to execute a shell command upon reboot.
   -ct : cron_time, Cron time settings to set when the shell command should be executed
   -sc : shell command to be executed

2. The second and primary component is the CronJob() class. This takes values from the argument parser, handles the different types of inputs and if the current time matches the cron time settings, runs the bash command at the beginning of a minute.

   The basis of this is that the current timestamp is taken, then checks are done against the input cron time settings, which are parsed on the go. If each time parameter matches the current time, the shell command is executed.

This is run initially at the beginning of a second, then has a sleeper of 59.5 seconds between each run as the lowest time granularity specified is per minute. Because of this, it assumes that our program runs within 0.5 seconds (500 milliseconds) which should be plenty. By doing so, we can significantly reduce the computational load required as the program is only executed for 500 milliseconds each second. The program is run until the user stops it with keyboard shortcut `ctrl+c`.

Limitations of Current Solution:

- @reboot is not handled currently

- Erroneous values simply raise a value error and could be handled better. We simply assume the user has entered values correctly (specifically the ordering and length of input). Missing values are not handled either.
    - There are also no sanity checks on the different inputs available

- Not enough testing has been done to ensure the robustness of our solution, especially for special cases where @day_of_week and @day_of_month are both specified

- Testing for how fast our solution runs has not been implemented.

- Code base requires much more refactoring and documentation (mentioned in all the #TODO's) to make it much more wieldable and portable (it looks atrocious and I have to

restrain from improving because of the self imposed time limit). This tool could be quite useful to others as an open source package but much work is required before then.
    - Large parts of the code base is repetitive and a lot of easy wins can be achieved. This is a definite TODO
    - A lot of the checks are 'if' statements and even something as simple as drafting a tree diagram can help visualise how our program runs as there are multiple possible break points

- Unsure if I captured all of Cron's functionalities, there may be many more I am unaware of
    - I have tried to replicate

Opportunities for Further Work:

- Design Improvement:
    - Parse arguments just once as they are static then store as variables so only time comparison checks are made
    - Modify time.sleep() to have a variable sleep time depending on the input, ex. If we set the cron runtime to run once a month, we can have time.sleep() to be set for a few seconds less than a month to be hardly computationally expensive at all!
    - Only runs on one thread at the moment, Implementation to spread out computation over multiple threads, or even GPU (requires investigation). Also can only schedule one thing at a time, it would be beneficial if we can schedule multiple jobs.
    - Can we take advantage of how Python loads variables into memory?

- Functionality:
    - Be able to set different time zones
    - Be able to run as a background task

- Usability:
    - Discuss with actual users who may be using this tool. A command line might not be so accessible to everyone and may want to use a GUI

- Diagnosis:
    - Being able to "check in" and monitor live scheduled tasks easily, as well as being able to cancel them at ease
    - Error and bug reporting

- Argparser Improvement:
    - Remove -r option from command line parser and be able to parse it as a cron_time argument

Hindsight:

- Spent too much time playing around with irrelevant aspects such as argument parser to try and make usability good for a proof of concept (version 1) and should have focused on something simple that works from the get-go
- Spent too much time trying to play with other libraries before implementing a custom solution. Whilst It should have occured to me sooner that these did not offer the full flexibility of what I had in mind and
- I set a time limit to the coding part of the task from 9am to 5pm, with about 2 hours total of breaks between and think that more of the allocated time should have been spent building a more robust solution with cleaner coding practices
    - That being said, I also had to restrain myself from adding more code after further thought following the time limit despite all the possible improvements in ‘further work’ above.
- Although it got slightly stressful around 3/4pm, it was overall a fun task to try and implement. I’m going to also try something new and create a Wikipedia account to suggest an edit for the Cron page so it is slightly easier to interpret.