INFO1112 2020 Assignment 1 (version 3 - 19th Oct)

Due Week 8, Friday 23rd October at 5pm

This assignment is worth 20% of your overall grade for the course.

Assessment

The assignment will be marked with an automatic testing system on Ed as well as manual inspection by a tutor. An mark will be given based on the tests passed (10%) and a mark will be given for overall style, quality, readability, etc (5%). You are expected to write your own tests and submit them with your code, and a mark will be given based on manual inspection of your tests (5%).

There will be public test cases made available for you to test against, but there will also be extra non-public tests used for marking. Success with the public tests doesn't guarantee your program will pass the private tests.

This assignment involves developing a system to run programs at scheduled times. This is similar to the Unix and Linux "cron" system.

Your program will read a configuration file that specifies what programs are to be run and when. You will be able to specify that a given program is run periodically at particular times, for example every Tuesday at 1pm, run a certain script. Alternatively you could specify that a program be run at 8am, 12noon, 2pm and 4pm everyday.

Your system will consist of two programs:

- A program (`runner.py`) that will run in the background, reading the configuration file that specifies what programs (with parameters) it should run and when they should be started. Normally `runner.py` would be started when the system is initialised, but for this assignment you can run the program as a background process..
- The second command (`runstatus.py`) that is designed to get the current status from `runner.py` and send it to the standard output.

Programs like `runner.py` are normally called "daemons" (pronounced the same as "demon") and run in the background. There are many examples of daemons in Linux, for example a printer daemon will manage a queue of files to send to the

printer. The configuration file for `runner.py` will be named `$HOME/.runner.conf` and contains a list of programs to start, what time to start them and if they should be run at that time regularly. The configuration file is described in more detail below. The `runner.py` program should keep information about the current status: what programs need to be run, what time they last ran, and when they are next due to be run. If `runner.py` receives the `SIGUSR1` signal it should open the file `$HOME/.runner.status` and write the status information in a human readable form to it, then close the file. This will be read by `runstatus.py`

At startup, `runner.py` should write its process ID into the file `$HOME/.runner.pid` and check that the status file `$HOME/.runner.status` exists. If the status file does not exist it should be created by `runner.py`.
Note that $HOME is a shell variable that holds the pathname of the home directory. You could also write the pathnames for the files as ~/.runner.pid or ~/.runner.status since the tilde (~) character also means the home directory.

The second command (`runstatus.py`) is designed to get the current status from the system. It should send the `SIGUSR1` signal to the `runner.py` program (using the PID stored in `$HOME/.runner.pid`) and then open and read the status file `$HOME/.runner.status` and send the contents to the standard output. Then close the status file, reopen it in write mode to truncate it to zero length, close it again and finally terminate. It should give an error message if anything fails, such as `$HOME/.runner.pid` or `$HOME/.runner.status` missing, or nothing appears in the file `$HOME/.runner.status` after 5 seconds. See error handling section below.

Configuration File

The configuration file for runner.py will contain one line for each program that is to be run. Each line has the following parts:

timespec program-path parameters

- *timespec* is the specification of the time that the program should be run
- *program-path* is a full path name of a program to run and the specified time(s)
- *parameters* are the parameters for the program.

The *timespec* has the following format:

[**every**|**on** *day*[,*day*…]] **at** *HHMM*[,*HHMM*…] **run**

Square brackets mean the term is optional, vertical bar means alternative, three dots means repeated. Terms in **bold** are keywords. Times are in 24 hour clock format in the range 0000-2359. Days are full day names that are case sensitive and start with capital letter. Space between elements can be any whitespace.

Examples:

```
every Tuesday at 1100 run /bin/echo hello
```
- every tuesday at 11am run "echo hello"

```
on Tuesday at 1100 run /bin/echo hello
```
- on the next tuesday only, at 11am run "echo hello"

```
every Monday,Wednesday,Friday at 0900,1200,1500 run
/home/bob/myscript.sh
```
- every monday, wednesday and friday at 9am, noon and 3pm run myscript.sh

```
at 0900,1200 run /home/bob/myprog
```
- runs /home/bob/myprog at 9am and noon only. If 9am is already passed then run tomorrow at 9am etc

Status Messages

Status messages are
- a line of text for each time a program has been run or there was an error
- a line showing the next time it will be run.

The format for the output lines is:

ran *date-time program-path parameters*
error *date-time program-path parameters*
will run at *date-time program-path parameters*

where words in bold are fixed, *date-time* is a string in python `time.ctime()` format, *program-path* is the full path of the program and *parameters* are the parameters used.

Error handling

Your program should do detailed error checking. In particular,

- if you cannot open the configuration file you should print "configuration file not found" and terminate.
- if an error is detected in the configuration file you should print "error in configuration: *line*" where *line* is the line of the file containing the error, then terminate
- if the configuration file is empty, you should print "configuration file empty"
- if an error occurs during the fork/exec of a program, this should be noted in the time record and the "error *date-time program-path parameters***"** message produced in the status. In this case runner.py does not terminate.
- if a file such as .runner.pid or .runner.status cannot be created or is not found, or has some other error you should print: "file *file-name error-message*" where *file-name* is the file that caused the error and *error-message* is an informative message. Then terminate.
- if the status does not arrive in 5 seconds print "status timeout" and terminate.
- if runner.py has no programs to run (this will eventually occur if there are no "every" lines in the configuration) then print the message "nothing left to run" and terminate.

Your programs should never crash. If an error occurs a suitable message or status produced. All error messages should be sent to stderr.

If any error is detected you need to report it. This includes failures of fork/exec, creating a file, not finding a file that should be there etc.

An experienced programmer checks for as many error conditions as they can. Thinking that "this can never happen therefore I won't check for it" is a bad idea.

You can assume processes finish inside 60 seconds and wait for each process to complete before running the next. [Note that this would not be acceptable for a real "cron" program. In order to handle all cases you would need to use the "waitpid" function that is non-blocking, and check regularly.] Also, you do not need to handle the case when two programs are to be run at exactly the same day and minute.

Please note, further clarifications to this specification may be posted on Ed and new editions of the specification published on Canvas.

Implementation

The assignment is to be implemented in two Python programs. A scaffold file will be provided. You are expected to write legible code following the PEP-8 style guide.

The only Python modules which you are allowed to import are os, sys, time, enum, stat, datetime, signal, collections and re. The os.system function is not permitted - use fork/exec and wait to run programs. If you want to use an additional module which will not trivialize the assignment, ask your tutor, and the allowed library list may be extended.

Testing

You are expected to write a number of test cases for your program. These should be simple input/output tests. An example test configuration file will be included in the scaffold. You are expected to test every execution path of your code.

You will need to test with different configuration files and show that your program can handle errors for incorrect input.

You should write a simple testing script in Bash (test.sh) to run your input/output tests to simplify and automate the testing process. This will be looked at by the marker but won't be used when the marker tests your code.

Below you will find some **correct** configuration lines to test runner.py. They are annotated to show what was being tested, but obviously the annotation should not go into the configuration file.

```
on Tuesday at 1100 run /bin/date
-- simple case
on Tuesday at 0000 run /bin/date
-- time 0000
every Tuesday at 1100 run /bin/date
-- every week on the same day
every Monday,Tuesday,Wednesday at 0900,1000 run /bin/date
-- multiple days, multiple times every week
on Tuesday at 0800 run /bin/echo Hello World!
-- more complex program arguments
```

Below you will find some **incorrect** configuration lines to test runner.py. They are annotated them to show what the error was but obviously the annotation should not go into the configuration file.

```
every Tuesday,Wednesday,Tuesday at 1200,1100 run /bin/date
--- repeated day
every Tues at 1100 run /bin/date
-- incorrect dayname
every Tuesday at 11000 run /bin/date
-- incorrect time
every tuesday at 1100 run /bin/date
-- incorrect dayname (case is wrong)
on Tuesday at 1100 /bin/date
-- no run keyword
on every Tuesday at 1100 run /bin/date
-- bad syntax
on Tuesday at 2400 run /bin/date
-- times range from 0000 to 2359
on Tuesday at 1100 run /bin/date
on Tuesday at 1100 run /bin/date
-- duplicate run time
on Tuesday at 1100 run
-- program path missing
on Tuesday at 1100 run /what/huh?
-- program not found (presumably)
on Tuesday at 1260 run /bin/date
-- incorrect time
on Tuesday at 123 run /bin/date
-- incorrect time
on Tuesday at 12-0 run /bin/date
-- incorrect time
on Monday at 1100,1200 run /bin/echo Hello
on Monday at 1000,1100 run /bin/echo there
-- duplicate time
```

Submitting your code

An Ed assessment workspace will be available for you to submit your code. You should create a git repository for your code and push it to the Ed Assessment using the link provided.

You should submit four files: runner.py, runstatus.py, test.sh and runner.conf. The test.sh is a shell script that should run your programs with a valid ~/.runner.con file containing a single correct configuration line. The runner.conf must contain any other test data you used to test your programs.

Test cases will be released until Monday 19th October. Additionally, there will be a set of unreleased test cases which will be run against your code after the due date.

Any attempt to deceive the marking system will be subject to academic integrity proceedings.

Academic Declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.