SCALE FOR PROJECT TASKMASTER (/PROJECTS/TASKMASTER)

Introduction

In order to have a productive and tolerable grading session, we ask that you:

- Stay courteous, polite, respectful and constructive during this session. The bond of trust between members of the 42 community depends on it;
- Take care to show the graded person(s) the problems you notice, and explain them as best you can;
- Accept that there may be differences in interepretation on the featureset and/or what the subject requires. Stay open-minded, try to honestly determine who is right and who is not, and grade accordingly.

Guidelines

Remember that you must ONLY grade what's on the turn-in repository !

You have to "git clone" the repository, and grade what's on it, AND ONLY WHAT IS ON IT.

Attachments

En - Subject (https://cdn.intra.42.fr/pdf/pdf/5814/taskmaster.en.pdf)

Preliminary checks

First and foremost

Check the following elements:

- There is something in the git repository
- The Makefile, if applicable, is present and has the required rules
- No unauthorized libraries are used. In this project, the students can use any language they want, and external libraries are only permitted for the sake of parsing the configuration and doing the client/server bonus. Other than that, the limit is the standard library.

If one of these elements is not in confirmity with what the subject requires, the session stops. You may still debate on the project, but you are not to grade the students.

During the rest of this session, if the program has an inappropriate behaviour (Segfault, bus error, double-free, uncaught exception, etc ...), the session stops.

☑ Yes ☒ No

Basic features

For this entire section, the students have to PROVE to you that what this grading scale asks is valid for their program. For example, if they cannot show you that programs are indeed restarted when they die, then they do not deserve points on the related question. Repeat, do NOT count a question as true if the students are not able to prove to you that it is. It is a requirement of the subject.

Control shell

The program offers some kind of control shell, either in the program itself or in a separate program (like supervisorctl).
The shell allows one to start, stop, and restart programs.

☑ Yes ☒ No

Configuration file

The program loads its running configuration from a configuration file, whatever the format. For this feature, external libraries

(like yaml parsing libraries, for example) are allowed. Programs described in the configuration file are actually loaded, their status can be accessed with the shell, etc...

☑ Yes ☒ No

Logging

There is a logging system that logs to a file or better (More points in the bonus section for this). For this to be true, there has to be a reasonable number of distinct events taken into consideration :
When programs are started, stopped, restarted, when they die unexpectedly, when the project aborts starting them because of too many retries, etc...

☑ Yes ☒ No

Hot-reload

The configuration can be reloaded while the program is running, both with a SIGHUP and with a shell command. When the reload takes effect, the running state of the program has to be altered to conform to the new configuration. However, programs that are not affected by the configuration change must not be restarted in any way (That would be very, very sloppy)

☑ Yes ☒ No

Configuration options

For this section, each question regards a specific configuration option that is required by the subject. The students must show you which option in their configuration scheme is concerned, and then show you that it works as requested.

Command used to launch the program

Command used to launch the program

☑ Yes ☒ No

Number of processes to start and keep running

Number of processes to start and keep running

☑ Yes ☒ No

Whether the program should automatically be started at launch or not

Whether the program should automatically be started at launch or not

☑ Yes ☒ No

Whether the program should restart always, never, or only on unexpected exit

Whether the program should restart always, never, or only on unexpected exit

☑ Yes ☒ No

Which return codes represent an "expected" exit status

Which return codes represent an "expected" exit status

☑ Yes ☒ No

How long to wait before deciding the program is "successfully started"

How long to wait before deciding the program is "successfully started"

☑ Yes ☒ No

How many times to attempt restarting the program before aborting

How many times to attempt restarting the program before aborting

☑ Yes ☒ No

Which signal should be used to gracefully stop the program

Which signal should be used to gracefully stop the program

Yes No

How long to wait after a graceful stop attempt before killing the program

How long to wait after a graceful stop attempt before killing the program

Yes No

Option to redirect stdout/stderr to a file or to discard them

Option to redirect stdout/stderr to a file or to discard them

Yes No

Environment variables to set for this program

Environment variables to set for this program

Yes No

Working directory for this program

Working directory for this program

Yes No

Umask for this program

Umask for this program

Yes No

Tests

Tests

Does the program stand up to a variety of tests ?

You should begin by trying the following:

- Kill a supervised process, and check that it does restart automatically
- Try supervising a process that only ever exits with an error, and check that it is indeed aborted after a set number of retries

You should then try any (reasonable) test you can think of to make the program behave erratically. If there is a conflict between you and the students about what constitutes a reasonable test, the one with the baseball bat is right.
If the program is still standing, then answer Yes. Otherwise... too bad !

☑ Yes ☒ No

Bonuses

Bonuses

Count 1 for each distinct, correctly implemented, and at least vaguely useful bonus feature.

Here are the ones suggested by the subject:

- Privilege de-escalation on launch (allows a program to run as a specific user)
- Client/server architecture like supervisor (Main program is a daemon that actually does the work, and a separate program communicates with it via unix/tcp sockets or another method to provide the control shell)
- More advanced logging/reporting facilities (Email, http, syslog, whatever)
- Option to "attach" a supervised process to the current

console, in the fashion of screen or tmux (Counts for 2 !).

Of course, any other bonus that the students decided to implement counts as well.

Rate it from 0 (failed) through 5 (excellent)

Ratings

Don't forget to check the flag corresponding to the defense

✔ Ok ★ Outstanding project

Empty work Incomplete work No author file Invalid compilation

Norme Cheat Crash Incomplete group Leaks

⊘ Forbidden function

Conclusion

Leave a comment on this evaluation

Preview!!!