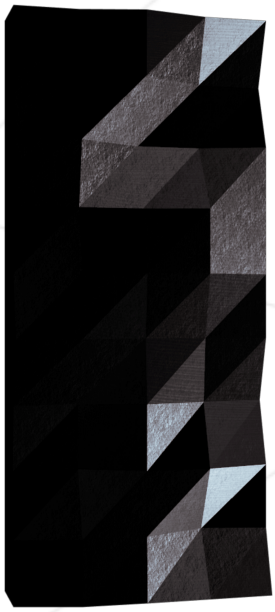

Taskmaster

Résumé: Le but de ce projet est de vous faire développer un "job control daemon", avec des features similaires à celui de supervisor.

Table des matières

Chapitre I

Préambule

Voici un petit texte à propos des `Dwellers` :

```
Picking a fight with a species as widespread, long-lived, irascible and (when
it suited them) single-minded as the Dwellers too often meant that just when
(or even geological ages after when) you thought that the dust had long since
settled, bygones were bygones and any unfortunate disputes were all ancient
history, a small planet appeared without warning in your home system,
accompanied by a fleet of moons, themselves surrounded with multitudes of
asteroid-sized chunks, each of those riding cocooned in a fuzzy shell made up
of untold numbers of decently hefty rocks, every one of them travelling
surrounded by a large landslide's worth of still smaller rocks and pebbles, the
whole ghastly collection travelling at so close to the speed of light that the
amount of warning even an especially wary and observant species would have
generally amounted to just about sufficient time to gasp the local equivalent
of "What the fu--?" before they disappeared in an impressive, if wasteful,
blaze of radiation.
```

Que sont les `Dwellers` ? Clique ici ! Non, plus sérieusement, allez lire `L'Algébriste`. Ce projet est bien plus simple si vous l'avez lu.

Chapitre II

Introduction

Unix est un système d'exploitation multi-tâche et multi-utilisateurs, ce qui veut dire d'une part que plusieurs individus peuvent accéder aux ressources de l'ordinateur en même temps et d'autre part que plusieurs programmes peuvent s'exécuter en même temps.

En même temps ? non, pas tout à fait... En effet, de nos jours, la majorité des ordinateurs ne compte encore qu'un seul processeur, lequel n'est capable d'exécuter qu'une seule instruction à un instant précis. Si nous désirons lancer plusieurs programmes en même temps, (nous parlerons ici de processus), ces derniers accèderont chacun à leur tour aux ressources de la machine. Une telle politique est dite de time sharing (partage du temps).

Bien entendu, un tel système n'est pas simple à gérer. En effet, il faut prévoir un partage équitable des ressources entre les divers utilisateurs, de manière, par exemple, à favoriser les processus interactifs (ceux qui communiquent et interagissent directement avec l'utilisateur, comme, par exemple, un éditeur de texte, ou un programme de dessin vectoriel) au détriment des programmes de calcul batch.

Chapitre III

Objectifs

Ce que vous devrez réaliser ici est un processus de job control à part entière. Vous pouvez trouver un bon example ici avec supervisor.

Pour rester dans quelque chose d'assez simple, votre programme ne tournera pas en tant que `root`, il ne sera pas obligatoire qu'il s'agisse d'un processus. Il sera executé via le terminal et fera son travail pendant qu'il donnera l'accès au shell à l'utilisateur.

Chapitre IV

Consignes générales

IV.1 Langages

Vous êtes libre d'utiliser le langage qu'il vous plaît. Les librairies pour le parsing des fichiers de configuration et, si vous décidez de les implémenter pour des bonus, un client et server. Vous êtes limités à la librairie standard de votre language.

IV.2 Evaluation

Pour l'évaluation soyez prêt à :

- Démontrer que votre programme implémente correctement chaque feature en le lançant avec un fichier de configuration que vous fournirez.
- Votre programme sera testé par votre correcteur, de façon diverses et variées dont tuer manuellement les processus supervisés, essayer de lancer des processus qui ne se démareront jamais correctement, lancer des processus qui génèrent énormement d'outputs, etc...

Chapitre V

Partie obligatoire

Votre programme doit être capable de démarrer des jobs comme des processus enfants, les laisser "en vie" et les relancer si nécessaire. Il doit aussi savoir à n'importe quel moment si ces processus sont en vie ou morts.

Les informations sur quel programme doit être lancé, comment, combien, s'ils doivent se relancer, etc... doivent être contenues dans un fichier de configuration. Vous avez le choix du format (le YAML peut être une bonne idée mais à vous de décider). Cette configuration doit être lancée au démarrage et doit être rechargeable pendant que `taskmaster` en est train de tourner en lui envoyant un SIGHUP. Lorsqu'il a rechargé, votre programme doit appliquer les changements sur son état courant (supprimer des programmes, en ajouter, changer leurs conditions de contrôle, etc...), mais il ne doit PAS supprimer les processus qui n'ont pas subit de changements avec le rechargement.

Votre programme doit avoir un système de registres afin d'enregistrer les évènements dans un fichier local (Lorsqu'un programme se lance, s'arrête, se relance, quand il meurt de manière innatendue, quand la configuration s'actualise, etc...)

Une fois votre programme lancé il doit rester actif en arrière-plan et donner l'accès à un shell à l'utilisateur. Il ne DOIT pas être un shell à par entière comme le `42sh`, mais il doit au moins avoir des fonctionnalités de base telles que l'édition de lignes, un historique, etc... l'auto-complétion serait aussi sympathique. Inspirez-vous du shell de `supervisor`, `supervisorctl`.

Ce shell devra au moins autoriser l'utilisateur à :

- Voir le status de tous les programmes décris dans le fichier de configuration (avec la commande "status")
- Lancer / arrêter / relancer les programmes
- Recharger le fichier de configuration sans que le programme principal s'arrête
- Arrêter le programme principal

Le fichier de configuration doit autoriser l'utilisateur à spécifier ce qui suit, pour chaque programme cela doit être supervisé :

- La commande à utiliser pour lancer le programme
- Le nombre de processus à lancer et laisser tourner
- Choisir de lancer ce programme au démarrage ou non
- Choisir si le programme doit toujours être relancé, jamais, ou uniquement lorsqu'il s'arrête de manière innatendue
- Quel code de retour represente une sortie "attendue" du programme
- Combien de temps le programme doit-il tourner après son démarrage pour que l'on considère qu'il s'est "lancé correctement"
- Combien de fois un redémarrage doit être réalisé avant de s'arrêter
- Quel signal doit être utilisé pour arrêter (i.e. exit gracefully) le programme
- Combien de temps d'attente après un graceful stop avant de kill le programme
- Options pour retirer les stdout/stderr du programme ou pour rediriger vers des fichiers
- Des variables d'environnement a set avant de lancer le programme
- Un répertoire de travail a set avant de lancer le programme
- Un umask a set avant de lancer le programme

Chapitre VI

Partie bonus

Nous vous encourageons à implémenter tout ce que vous pensez être utile à votre projet. Vous receverez des points pour cela si c'est correctement implémenté et un minimum utile.

Voici quelques idées :

- Une système d'élévation des privilèges au lancement (Devra être lancé en tant que root, vous aurez donc besoin d'une VM pour cela...)
- Une architecture client/serveur pour permettre deux programmes separes : un "daemon", ce que fait l'actuel "job control", ainsi qu'un programme de controle, qui fournit un shell pour l'utilisateur, et communique avec le "daemon" over UNIX or TCP sockets. (tres semblable a `supervisord` et `supervisorctl`)
- Un système plus avancé de logs et de reports (avertissements via email/http/syslog/etc...)
- Autoriser l'utilisateur à "attach" un processus supervised à sa console, comme `tmux` ou `screen` le font, puis les "detach" a partir de celui ci et les refaire tourner en arrière plan.

Chapitre VII

Annexes

VII.1 Exemple de fichier de configuration

Voici a quoi pourrait ressembler un fichier de configuration pour votre `taskmaster` :

```yaml
programs:
    nginx:
        cmd: "/usr/local/bin/nginx -c /etc/nginx/test.conf"
        numprocs: 1
        umask: 022
        workingdir: /tmp
        autostart: true
        autorestart: unexpected
        exitcodes:
            - 0
            - 2
        startretries: 3
        starttime: 5
        stopsignal: TERM
        stoptime: 10
        stdout: /tmp/nginx.stdout
        stderr: /tmp/nginx.stderr
        env:
            STARTED_BY: taskmaster
            ANSWER: 42
    vogsphere:
        cmd: "/usr/local/bin/vogsphere-worker --no-prefork"
        numprocs: 8
        umask: 077
        workingdir: /tmp
        autostart: true
        autorestart: unexpected
        exitcodes: 0
        startretries: 3
        starttime: 5
        stopsignal: USR1
        stoptime: 10
        stdout: /tmp/vgsworker.stdout
        stderr: /tmp/vgsworker.stderr
```

VII.2 Trying out supervisor

`supervisor` est disponible sur PyPI comme un package de Python. Pour l'essayer, la methode la plus simple est de faire un `virtualenv` sur votre home, activez le, et installez le `supervisor` avec `"pip install supervisor"`. Vous devez installer `python` avant, il est disponible sur `Homebrew`.

Vou pouvez alors faire un fichier de configuration pour gerer un ou deux programmes, lancez `supervisord -c myconfigfile.conf`, puis interagissez avec pour utiliser `supervisorctl`.

Gardez a l'esprit que `supervisor` est mur, riche en fonctionnalites, et que ce vous devez faire avec `taskmaster` est moins abouti, vous devez juste le voir comme une source d'inspiration. Par exemple, `supervisor` offre le controle du shell dans differents process qui communique avec le programme principal via un "UNIX-domain socket", tant que vous etes le seul a assurer le controle du shell dans le programme principal.

Si vous avez un doute sur le comportement de votre programme dans certaines situations, ou quelle fonctionnalite donner a certaines options... Bien, quand vous avez un doute, reproduisez le comportement de `supervisor`, ainsi vous ne pourrez pas avoir tort.

Chapitre VIII

Rendu et peer-évaluation

Rendez-votre travail sur votre dépot GiT comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance.

Bon courage à tous et n'oubliez pas votre fichier auteur !