Projet

1 Spécifications

L'objectif de ce projet est de créer une commande `detecter` pour lancer périodiquement un programme et détecter les changements d'état. Votre commande doit admettre les arguments suivants :

`detecter` [`-t` *format*] [`-i` *intervalle*] [`-l` *limite*] [`-c`] *prog arg ... arg*

Les options sont les suivantes :

- l'option `-i` donne l'intervalle (en millisecondes) entre deux lancements du programme (valeur par défaut : 10 000 millisecondes) ;
- l'option `-l` donne la limite du nombre de lancements, ou 0 pour aucune limite (valeur par défaut : 0, c'est-à-dire pas de limite)
- l'option `-c` détecte également les changements de code de retour, c'est-à-dire l'argument de `exit` du programme appelé (valeur par défaut : pas de prise en compte du code de retour) ;
- l'option `-t` provoque l'affichage de la date et l'heure de chaque lancement, avec le format spécifié, compatible avec la fonction de bibliothèque `strftime` (valeur par défaut : aucun affichage).

La commande `detecter` lance le programme spécifié par *prog* avec les arguments suivants. Au premier appel, la sortie standard de la commande est affichée (ainsi que son code de retour si l'option `-c` est activée). Après chaque appel, la sortie standard (ainsi qu'éventuellement le code de retour) est analysée et, en cas de différence avec la sortie précédente, la nouvelle sortie standard est affichée.

2 Exemple

Dans l'exemple suivant, le programme `ls` est appelé avec les arguments `-l` et `/tmp` 4 fois, avec un intervalle de 10 secondes entre deux appels et l'affichage de l'heure à chaque appel. À chaque fois que la sortie standard de la commande change, le nouveau contenu est affiché :

```
turing> ./detecter -c -i 10000 -l 4 -t "%H:%M:%S" ls -l /tmp
14:47:21
total 3
--w------- 1 pda       GG_MAI  5164 Jan 01 14:45 krb5cc_41866
--w------- 1 montavont GG_MAI  5204 Jan 01 14:45 krb5cc_76543
drwx------ 2 pda       GG_MAI  4096 Jan 01 15:10 ssh-BkYk86pgQhHy
exit 0
14:47:31
14:47:41
total 2
--w------- 1 pda       GG_MAI  5164 Jan 01 14:45 krb5cc_41866
drwx------ 2 pda       GG_MAI  4096 Jan 01 15:10 ssh-BkYk86pgQhHy
14:47:51
turing>
```

Dans cet exemple, on constate que lors de la première exécution de `ls`, le résultat initial est affiché : 3 fichiers sont trouvés et le code de retour est nul. Puis, rien ne se passe pendant au moins 10 secondes (la deuxième itération ne détecte aucun changement). Lors de la troisième itération, à 14h47 et 41 secondes, un changement est détecté car un fichier a été supprimé. Le code de retour n'ayant pas été modifié, il n'est pas ré-affiché. Lors de la quatrième et dernière itération, aucun changement n'est à nouveau détecté.

3 Contraintes particulières et simplifications

Pour simplifier l'implémentation, en cas d'erreur, on arrêtera l'exécution avec un message approprié. De même, on ignorera la sortie d'erreur standard du programme lancé. Une sortie du programme pour une raison autre qu'un appel à `exit` sera considérée comme une erreur.

Pour réaliser l'attente entre deux itérations, vous utiliserez la fonction de bibliothèque `usleep`. Vous utiliserez la fonction `getopt` pour analyser les options. Attention toutefois : l'implémentation de `getopt` sur Linux ne respecte pas la norme POSIX[1], on mettra donc un « + » en premier dans la chaîne d'arguments (troisième argument de `getopt`) comme indiqué dans le manuel Linux.

Vous respecterez scrupuleusement le modèle d'affichage utilisé dans l'exemple, de manière à ce que les tests puissent être automatisés (voir les jeux de tests disponibles sur Moodle). Vous n'utiliserez aucun fichier intermédiaire. Bien sûr, vous n'utiliserez que les primitives systèmes, sauf pour l'allocation mémoire, les affichages, l'analyse des options, l'attente et la mise en forme de l'heure. Pour ce projet, on assimilera `execvp` à une primitive système.

4 Travail demandé

On demande de réaliser :

- la commande `detecter` ;
- des jeux de tests complémentaires et la mesure de la couverture de code, pour vérifier que votre programme est bien conforme aux spécifications ;
- un rapport décrivant les élements demandés précédemment.

Votre implémentation doit fonctionner sur `turing`. Vous vous attacherez à respecter les spécifications et à le prouver par vos jeux de tests, ainsi qu'à veiller à ne pas avoir de fuite de mémoire.

Vous noterez qu'une archive est fournie sur Moodle, contenant un fichier `Makefile` et des jeux de test.

5 Modalités de remise

Le projet est à réaliser par groupes de deux étudiants. Votre projet doit contenir :

- les fichiers sources ;
- le rapport au format PDF, contenant en particulier les mesures demandées ;
- l'ensemble des jeux de tests et la mesure de couverture obtenue avec `gcov`.

Vous déposerez votre projet, débarrassé de tout fichier binaire autre que votre rapport, sous forme d'une archive au format *tar.gz* sur Moodle dans l'espace prévu à cet effet, avant le dimanche 9 avril 2017 à 20 h.

1. L'implémentation Linux d'origine GNU prend l'initiative (malheureuse) de réordonner d'office les arguments pour que toutes les options soient mises avant les arguments, ce qui fait que dans l'exemple, le `-l` de `ls` est déplacé avant `ls`.