Projet : findexec

1 Introduction

Ce projet consiste à implémenter une version parallèle de la commande :
`find <rep> -type f -exec <commande> {} ;`

Le programme que vous allez écrire s'exécute de la manière suivante :

```
findexec -n 10 -e wc /home /tmp
```

Il explore les fichiers des arborescences indiquées (ici `/home` et `/tmp`) et applique une commande (ici `wc`) sur chacun des fichiers réguliers (normaux) trouvés. L'option `-n 10` indique qu'on peut lancer au plus 10 processus en parallèle, c'est-à-dire que si 10 processus sont en cours d'exécution et qu'on veut en lancer un onzième, il faut attendre au préalable que l'un (quelconque) des 10 processus en cours soit terminé.

2 Arguments

Les arguments optionnels sont :
- `[-n <nombre de processus>]` nombre maximum de processus en cours d'exécution, par défaut : 1.
- `[-e <commande>]` nom de la commande à exécuter. Si cette option n'est pas donnée, le programme affichera simplement le nom (complet) du fichier trouvé et continuera.
- `[-h]` affiche une aide à l'utilisation du programme et quitte.

Vous utiliserez la fonction de bibliothèque `getopt(3)` pour lire ces arguments optionnels passés au programme.

Les arguments suivants constituent la liste des fichiers ou répertoires à parcourir pour exécuter la commande.

3 Indications sur l'implémentation

Vous pouvez écrire les fonctions suivantes :

1. `void attendre (void)`
   attendre qu'un des processus fils se termine. On considérera comme une erreur le cas où le processus ne se termine pas avec un code de retour nul.
2. `int lancer (const char *cmd, const char *fichier, int maxproc, int nproc)`
   lance la commande indiquée avec comme argument le fichier dont le chemin est fourni, en redirigeant en outre l'entrée standard de la commande sur le fichier.
   Le paramètre `maxproc` spécifie le nombre maximum de processus simultanés, et le paramètre `nproc` indique le nombre de processus actuellement en cours d'exécution. Vous utiliserez la primitive `execlp`. Votre fonction doit retourner le nouveau nombre de processus en cours d'exécution.
3. `int parcourir (const char *cmd, const char *racine, int maxproc, int nproc)`
   parcourt récursivement l'arborescence spécifiée par `racine` et lance la commande indiquée par `cmd` sur tous les fichiers rencontrés, tout en respectant le nombre maximum de processus indiqué. Cette fonction retourne le nombre de processus en cours d'exécution.
4. Écrivez enfin la fonction `main` correspondant aux spécifications ci-dessus. Vous n'oublierez pas d'attendre tous les processus encore en cours d'exécution à la fin.

Sauf contre-indication, vous utiliserez les primitives systèmes (on assimilera ici les fonctions de la famille `opendir` à des primitives systèmes). Pour les affichages, vous pourrez utiliser les fonctions de bibliothèque.

4 Gestion des erreurs

Vous prendrez un soin particulier à traiter toutes les erreurs qui peuvent survenir :

- lorsque les arguments du programme sont incorrects, un message d'aide est affiché et le programme renvoie un code de retour non nul ;
- lorsqu'un fichier spécial est rencontré, ou qu'un répertoire ne peut pas être lu, votre programme affichera un message d'avertissement sur `stderr`, mais continuera son exécution et le code de retour final sera non nul ;
- lorsque la commande lancée échoue pour une raison quelconque, il affichera un message d'erreur et s'arrêtera, en renvoyant un code de retour non nul ;
- lorsque tous les processus lancés se sont bien terminés, le code de retour sera 0.

Vous trouverez sur Moodle un `Makefile` de test de votre programme. Il crée un exécutable à partir du fichier source `.c` unique se trouvant dans le répertoire courant, puis crée une hiérarchie de fichiers (sous `./testdir/`) et lance plusieurs variantes de l'exécutable sur des parties de cette hiérarchie.

Ces tests ne sont pas exhaustifs (par exemple, il n'y a pas de fichier spécial ou de répertoire illisible, et les fuites mémoire ne sont pas détectées), mais ils vérifient que votre programme fonctionne conformément à l'énoncé dans plusieurs cas intéressants. Si `make` échoue à une certaine étape vous pouvez vérifier la raison de l'erreur et/ou la rejouer à la main : les commandes exécutées sont affichées en clair dans le terminal. N'hésitez pas à enrichir ce `Makefile` de nouveaux tests pour qu'il soit plus complet.

5 Modalités de remise

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

- les fichiers sources ;
- un rapport au format PDF contenant une description de votre implémentation, le résultat de jeux de tests, l'analyse de l'absence de fuite mémoire, et la mesure de couverture obtenue avec `gcov`.

Vous déposerez votre projet, débarrassé de la hiérarchie de test et 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 mercredi 23 novembre 2016 à minuit.