Projet Programmation système

Le but du projet est d'implémenter un outil de synchronisation de répertoires. Cet outil sera implémenté comme une commande

synchroniser options *destination source*

où `destination` est le répertoire de destination et `source` le répertoire source. Dans le cas le plus simple, sans option, `synchroniser destination source` parcourt le répertoire `source` et pour chaque fichier ordinaire dans le répertoire `source`, la commande `synchroniser` détermine s'il y a un fichier ordinaire du même nom dans le répertoire `destination`. Si c'est le cas, le programme compare la date de dernière modification des deux fichiers. Si la date de dernière modification d'un fichier dans le répertoire `source` est plus récente que la date de dernière modification de ce fichier dans le répertoire `destination`, alors le fichier sera recopié de `source` vers `destination`.

Remarque 1. Pour savoir quelle date est la plus récente, il suffit de comparer les champs `st_mtime` de la structure `struct stat` pour les deux fichiers; si `st_mtime` pour le fichier dans le répertoire `source` est supérieure à `st_mtime` pour le même fichier dans le répertoire `destination`, alors ce fichier dans `source` a été modifié plus récemment que le fichier correspondant du répertoire `destination`.

Remarque 2. Il y a cependant un petit problème à régler. Si on copie le fichier `toto.txt` du répertoire `source` vers le répertoire `destination`, alors la date de dernière modification de `destination/toto.txt` sera la date courante au moment où on a fait la copie. Si maintenant on échange les rôles de `source` et `destination` pour faire une synchronisation `synchroniser source destination`, alors le fichier `destination/toto.txt` sera copié vers `source/toto.txt`, ce qui est inutile. La solution consiste à modifier la date de dernière modification de `destination/toto.txt`. Après avoir copié `source/toto.txt` vers `destination/toto.txt`, le programme `synchroniser` mettra comme date de dernière modification de `destination/toto.txt` la date de dernière modification de `source/toto.txt`.

Les deux dates `st_atime` et `st_atime` sont modifiables à l'aide de la fonction

```
#include <sys/types.h>
#include <utime.h>
int utime(const char *chemin, struct utimbuf *temps)
```

La structure `struct utimbuf` possède deux champs: `time_t actime` et `time_t modtime`, `actime` est la date de dernier accès (sans intérêt pour nous) et `modtime` la date de dernière modification.

Les options

-r	on synchronise récursivement les sous-répertoires de `source` et `destination`, les sous-répertoires de sous-répertoires etc.
-n	sans option `-n`, on synchronise uniquement les fichiers qui existent dans les deux répertoires. Avec l'option `-n`, on copie aussi tout fichier qui existe dans `source` mais pas dans `destination` (avec `-r` on le fait aussi pour les sous-répertoires.)
-i	avec l'option `-i`, `synchroniser` demandera une confirmation de l'utilisateur pour chaque fichier que le programme s'apprête à synchroniser.
-s	avec l'option `-s`, on synchronise aussi les liens symboliques. Par exemple si `source` et `destination` contiennent un lien symbolique `toto` et le contenu du lien dans la source est `"../bin/toto"` alors, après la synchronisation, le contenu du lien `toto` dans `destination` sera le même. Par contre, on ne synchronisera pas le fichier `../bin/toto` pointé par le lien.

Il est possible de spécifier plusieurs options, par exemple `synchroniser -r -n destination source` synchronise récursivement les répertoires `source` et `destination` mais aussi recopie récursivement vers `destination` tout fichier et répertoire qui existent dans `source` mais n'existent pas dans `destination`. Cela peut entraîner la création de nouveaux sous-répertoires dans `destination`. Par exemple si `source` possède un sous-répertoire `bin` et `destination`n'a pas de sous-répertoire de même nom alors `synchroniser -r -n destination source` créera un sous-répertoire `bin` dans `destination` qui sera une copie de `source/bin`. Autrement dit, après l'exécution de la commande les arborescences avec les racines `destination` et `source` seront identiques (sauf pour les fichiers

spéciaux).

Remarques générales.

Le projet doit être convenablement divisé en fonctions et en plusieurs fichiers sources et accompagné d'un `Makefile`.

`Makefile` doit permettre la compilation de toutes les commandes avec un `make`.

La compilation de tous les fichiers sources doit être réalisée avec l'options **`CFLAGS = -Wall -pedantic`** sans aucun message d'avertissement. Il est possible qu'il faut ajouter aussi l'option `-std=c99` pour que votre code compile. Ces options concernent uniquement le compilateur `gcc`. Sur nivose il y a aussi le compilateur `cc` qui est différent de gcc, en particulier en ce qui concerne les options. Mais de toute façon le compilateur `cc` sur nivose peut poser de problèmes pour ce projet. Par contre il est possible d'utliser le compilateur `c99` sur nivose pour le projet (mais sans `-Wall -pedantic`).
De plus comme la première ligne de fichiers source (avant tout include) il convient d'ajouter
`#define _POSIX_C_SOURCE 200112L` ou
`#define _POSIX_C_SOURCE 200112`
Au lieu d'ajouter cette ligne on pourra passer cette constante à la compilation:

```
CFLAGS = -Wall -pedantic -std=c99 _D_POSIX_C_SOURCE=200112
```

`make clean` doit supprimer tous les fichiers `*.o` et les exécutables.

Dans les fichiers sources, il faut au moins commenter

- les fonctions en indiquant ce qu'elles sont censées faire et
- les variables globales (au niveau 0).

Modalités

Le projet sera réalisé en binôme ou monôme. Dans ce dernier cas, la notation tient compte du fait que vous avez réalisé votre projet seul(e).

La date limite de remise est **le 2 janvier 2013 à 23h59**.

La date de soutenance sera fixée ultérieurement.

Pour délivrer le projet, on crée un répertoire contenant les fichiers sources (`*.c` et `*.h`), le `Makefile` et le fichier `alire.txt`.

Dans le fichier `alire.txt`, il faut mettre les noms des auteurs et indiquer ce que vous avez réussi à faire (quelle options implémentées) et, éventuellement, ce qu'il reste encore à faire par rapport au travail demandé.

Le répertoire avec le projet doit être compressé sous forme `xxx_yyy.tar.gz`
où xxx et yyy sont vos noms. Quand on lance `tar xzvf xxx_yyy.tar.gz` pour décompresser votre archive, on doit obtenir le répertoire nommé `xxx_yyy` contenant votre projet. Le fichier d'archive avec votre projet est à déposer sous didel (sigle de cours SYSTEME 2012). Tout retard et le non respect de ces consignes seront pénalisés.