Langage C et Programmation Système
TP n° 8 : Inœuds, temps, tubes nommés, *locks*

Exercice 1 : Horodatage ***Métadonnées SF, date et heure, normes***

Dans cet exercice, vous allez écrire un programme `mytouch`, une version simplifiée de l'outil POSIX `touch`, qui change l'horodatage de la dernière modification, ou du dernier accès, ou des deux. La date et l'heure peuvent être spécifiées avec l'option paramétrée `-d date_time`. Dans le cas contraire, `mytouch` utilise la date et l'heure courante.

Synopsis :

```
mytouch [-am] [-d date_time] FILE [FILE]...
```

Options :

`-a` Modifie seulement l'horodatage du dernier accès au fichier. Ne modifie pas celui de la dernière modification, sauf si l'option `-m` est aussi présente.

`-d date_time` Le paramètre `date_time` est dans le format `YYYY-MM-DDThh:mm:ss`, où :
- `YYYY` est l'année ;
- `MM` est le mois `[01-12]` ;
- `DD` est le jour `[01-31]` ;
- `T` est la lettre T choisie comme séparateur ;
- `hh` est l'heure `[00-23]` ;
- `mm` est la minute `[00-59]` ;
- `ss` est la seconde `[00-60]`. (`60` correspond à une seconde intercalaire.)

`-m` Modifie seulement l'horodatage de la dernière modification du fichier. Ne modifie pas la celui du dernier accès, sauf si l'option `-a` est aussi présente.

1. Implémentez une version sans option qui modifie les deux horodatages. Pour chaque argument `FILE` de `mytouch` :

1. Si le ficher n'existe pas (*Indice :* `access`) :
   (a) Appelez la fonction `creat()` avec les arguments spécifiés ci-dessous :
   - L'argument `pathname` correspond à l'argument `FILE` courant (de `mytouch`).
   - L'argument `mode` est la valeur du "ou" inclusif bit à bit de `S_IRUSR`, `S_IWUSR`, `S_IRGRP`, `S_IWGRP`, `S_IROTH` et `S_IWOTH`.

   (b) Appelez la fonction `futimens()` avec les arguments spécifiés ci-dessous :
   - L'argument `fd` est le descripteur du ficher ouvert dans la question 1a.
   - L'argument `times` correspond à un tableau de `struct timespec`, dont les deux premiers éléments spécifient les horodatages d'accès et de modification, respectivement. (Cf. la page de manuel de `futimens` pour plus de détails.) Dans un premier temps, on mettra cet argument à `NULL`. (Pourquoi ?)

   (c) Fermez le fichier ouvert dans la question 1a.
2. Si le ficher existe, appelez la fonction `utimensat()` avec les arguments spécifiés ci-dessous :
   (a) L'argument `dirfd` est la valeur spéciale `AT_FDCWD`.

(b) L'argument `pathname` correspond à l'argument `FILE` courant (de `mytouch`).

(c) L'argument `times` correspond à un tableau de `struct timespec`, dont les deux premiers éléments spécifient les horodatages d'accès et de modification, respectivement. (Cf. la page de manuel de `utimensat` pour plus de détails.) Dans un premier temps, on mettra cet argument à `NULL`.

(d) L'argument `flags` est `0`.

2. Rajoutez la gestion des options `-a` et `-m`.

3. Rajoutez la gestion de l'option `-d date_dime`. (*Indices :* `strptime`, puis `mktime`.)

4. Regardez quelle partie de la norme POSIX vous venez de réaliser : `http://pubs.opengroup.org/onlinepubs/9699919799/utilities/touch.html`. Félicitez-vous.

Exercice 2 : Démon de compression *Tubes nommés et lock (utilisateur)*

Dans cet exercice, vous allez réaliser un service de compression de fichier, implémenté dans un script Bash. (On n'utilisera pas le langage C dans cet exercice.)

1. Créez un tube nommé `compresseur.fifo`. (*Indice :* `mkfifo`.)

2. Lancez un processus qui lit dans le tube et comprime ce qu'il lit avec `gzip`, et redirigez sa sortie dans un fichier `comprime.gz`. Vu que le processus attendra de lire quelque chose, il faudra soit le lancer en arrière-plan, soit utiliser deux terminaux virtuels. Vérifiez son comportement. (*Indices :* `gzip`, `<`, `>`, `&`.)

3. Créez un script Bash `gzipd.sh` qui :

1. vérifie la presence d'un tube nommé `compresseur.fifo` dans `/tmp`. Si ce tube n'existe pas, le script le crée ;
2. fait exactement ce qui est décrit dans la question 2, mais dans une boucle infinie, en suffixant le nom des fichiers générés avec la date et l'heure courante. (*Indices :* `while` *ou* `for`, *et* `date`.)

Testez votre script.

4. Analysez le comportement de `gzipd.sh` quand deux processus redirigent leurs sortie vers le même tube en même temps. (*Indice : concurrence critique.*)

5. Écrivez un deuxième script `gzipc.sh` qui prend en argument le chemin d'un fichier à comprimer, acquiert le *lock* sur le tube, et après redirige le contenu du fichier dans le tube. (*Indice :* `flock`.)

Exercice 3 : Chat *Tubes nommés et lock (système)*

Dans cet exercice, vous allez écrire une version primitive de l'outil de chat `talk`, qui permet à deux utilisateurs connectés sur la même machine de bavarder. Chacun des deux utilisateurs lance deux petits logiciels : un pour écrire (`talk`), l'autre pour lire (`listen`). Dans une phase d'initialisation :

1. On crée dans `/tmp` deux tubes nommés `user1` et `user2`, un pour chaque utilisateur, en donnant les permissions de lecture et d'écriture à tout le monde ;
2. Les deux tubes sont ouverts :
   - $\texttt{talk}_1$ (l'instance de `talk` du premier utilisateur) ouvre `user2` en mode `O_WRONLY`.

- `talk`$_2$ (l'instance de `talk` du second utilisateur) ouvre `user1` en mode `O_WRONLY`.
- `listen`$_1$ ouvre `user1` en mode `O_RDONLY`.
- `listen`$_2$ ouvre `user2` en mode `O_RDONLY`.

Après, les quatre processus (`talk`$_1$, `listen`$_1$, `talk`$_2$, `listen`$_2$) bouclent jusqu'à une interruption explicite due à l'utilisateur (par exemple avec `Ctrl+D`). À chaque tour, `talk`$_1$ (respectivement `talk`$_2$) :

1. lit une ligne l de son entrée standard,
2. écrit l dans `user2` (respectivement `user1`),
3. dort pour 0,2 secondes.

À chaque tour, `listen`$_1$ (respectivement `listen`$_2$) :

1. lit une ligne l de `user1` (respectivement `user2`),
2. écrit l sur sa sortie standard,
3. dort pour 0,2 secondes.

Dans la phase finale, les deux tubes `user1` et `user2` sont effacés.

1. Implémentez (en C) une version de base de `talk` et `listen` en supposant que :
- les deux programmes prennent en argument un chiffre permettant d'identifier l'utilisateur (`1` ou `2`) ;
- la création et la suppression des tubes sont effectuées par l'instance `talk`$_1$.

(*Indices :* `mkfifo`, `chmod`, `unlink`.)

2. Concevez une politique de gestion concurrente de ressources critiques et raffinez l'implémentation précédente pour que :
- les phases d'initialisation et de conclusion soient gérées par le premier processus exécuté, sans que l'on ait d'arguments à préciser ;
- une éventuelle troisième instance de `talk` ou de `listen` termine immédiatement avec une erreur.

(*Indice :* la page `man` de `fcntl`, en particulier `F_SETLK`, `F_SETLKW`, et `F_GETLK`.)