Programmation système avancée
TP n° 2 : Fichiers partagés entre processus

Exercice 1 :

1. Écrivez un programme qui ouvre un fichier `compteur.dat`, lit un entier de quatre octets en format gros-boutiste, ajoute 1 au compteur, puis écrit la nouvelle valeur dans `compteur.dat`. Si le fichier `compteur.dat` n'existait pas, votre programme y stockera la valeur 1.
2. Modifiez votre programme pour qu'il créé un processus fils qui incrémente le compteur 10^5 fois, puis attend la fin du processus fils, puis affiche la valeur du compteur. À l'aide de la commande `time` mesurez le temps d'exécution du programme, notez le résultat.
3. Modifiez votre programme pour qu'il crée 10 fils dont chacun incrémente le compteur 10^4 fois (il faudra prendre soin d'attendre la mort des 10 fils). Que constatez-vous ?
4. Modifiez votre programme pour protéger chaque mise à jour du fichier partagé par un *lockfile* nommé `compteur.dat.lck` (il faudra prendre le lock avant chaque mise à jour, à l'intérieur de la boucle) ; en cas de conflit, votre programme attendra 1 ms avant de réessayer. Mesurez les performances de votre programme à l'aide de `time`. Que constatez-vous ? Que se passe-t-il si vous interrompez votre programme à l'aide de *Ctrl-C* ?
5. Modifiez votre programme pour qu'il utilise un appel à `flock` (de type `LOCK_EX`) au lieu de créer un *lockfile*. Les performances ont-elle changé ? Que se passe-t-il si vous interrompez le programme ?
6. Dans la question précédente, aurait-il été correct de prendre un *lock* partagé pendant la lecture, puis le remplacer par un *lock* exclusif ? (Indication : c'est incorrect sur tous les systèmes, mais pas toujours pour la même raison.)

Exercice 2 :

Écrivez un programme qui crée un fichier `data.dat`, prend un *lock* exclusif sur le fichier, puis y écrit la chaîne « `Głaszcz jeździcze rżącego źrebca!` » suivie d'un caractère de fin de ligne. Il se suspend ensuite indéfiniment (sans relâcher le *lock*), par exemple à l'aide de l'appel système `pause`. Pendant que votre programme s'exécute, modifiez le fichier à l'aide de votre éditeur de textes favori. Que constatez-vous ?

Exercice 3 :

On se propose d'écrire un programme qui manipule un agenda. L'agenda sera stocké dans un fichier agenda.dat qui aura la forme suivante :

```
1707116400 Envoyer le TP2.
1707309900 Assurer le TP2.
```

Le fichier est constitué d'une suite triée d'événements, un par ligne. Chaque événement est représenté par sa date en secondes dans l'ère UNIX (comme la valeur retournée par `time`), suivi d'une description de l'événement.

1. Écrivez un programme `affiche-agenda` qui affiche le contenu de l'agenda sous une forme lisible par un être humain (vous pourrez vous servir des fonctions strftime et localtime) en prenant un *lock* partagé à l'aide de l'appel système `flock`. Deux instances de affiche-agenda peuvent-elles accéder au fichier en même temps ? Vérifiez votre hypothèse en ajoutant un appel à sleep au bon endroit.
2. Modifiez votre programme `affiche-agenda` pour que lorsqu'il est invoqué avec l'option -1 il n'affiche que le prochain événement à venir.
3. Écrivez un programme `ajoute-evenement` qui ajoute un événement passé en paramètre de ligne de commande. Vous pourrez utiliser la fonction `strptime` pour analyser la date passée en ligne de commande. Comme l'agenda doit être trié après l'exécution, il faudra lire le fichier entier, puis le récrire avec le nouveau événement. Utilisez le bon *lock*. Votre programme peut-il s'exécuter en même temps que `affiche-agenda` ?
4. Que se passe-t-il si votre programme est interrompu pendant l'exécution ? Modifiez-le pour qu'il ne corrompe pas l'agenda dans ce cas.