INF3173-232 TP1

On vous demande votre aide pour mettre en place un format de fichier pour une nouvelle application. En particulier, on veut conserver un numéro de version et la date de dernière modification dans les méta-données du fichier, en plus du contenu lui-même.

Le format de fichier binaire est le suivant:

- Le mot magique 0x1001CAFE (int, 4 octets)
- Le numéro de version sous forme d'un entier non-signé (unsigned long, 8 octets)
- Une estampille de temps de la dernière modification (struct timespec, 16 octets)
- Des données précieuses de taille quelconque

Voici la marche à suivre pour incrémenter le numéro de version:

- Ouvrir le fichier.
- Vérifier le mot magique. S'il ne correspond pas, alors on doit quitter avec une erreur pour ne pas corrompre un fichier d'un autre format.
- Ensuite, on lit la version actuelle, on l'incrémente, on écrit la nouvelle version dans le fichier.
- Inscrire une estampille de temps obtenu avec `clock_gettime()`.
- On ferme le fichier.
- Attention: il ne faut PAS effacer les données qui suivent, car c'est le contenu utile du fichier.
- Si le fichier n'existe pas, il faut le créer et inscrire une version zéro, et le temps actuel.

Cette fonction est appelée dans l'utilitaire `bumpvers`, ce qui incrémente la version dans le fichier spécifié. Le programme accepte un seul argument, soit le nom du fichier. Exécutez le programme et vérifiez le contenu du fichier. Voici un exemple de l'appel:

```
./bumpvers fichier.bin
```

La dernière étape consiste à réaliser un banc d'essai pour mesurer la différence de performance entre incrémenter la version dans le fichier avec un simple appel de fonction, ou faire l'incrément de la version avec l'utilitaire `bumpvers`. Pour cela, il faut compléter le fichier `benchmark.c` pour chaque mode. Pour le lancement du programme, vous devez utiliser `execlp()` ou `execvp()`. Le banc d'essai doit prendre 2 arguments, soit la méthode et le nombre de répétitions à exécuter. Le programme doit afficher sur la sortie standard le temps moyen écoulé pour exécuter chaque appel. Vous devez utiliser la fonction `clock_gettime()` pour obtenir le temps écoulé. Voici un exemple d'exécution du banc d'essai avec les arguments:

```
# Appel de fonction et effectuer 1000 répétitions
./benchmark fonction 1000
# Appel du programme bumpvers 10 fois
./benchmark commande 10
```

Vous pouvez ensuite comparer la performance entre les deux méthodes. On s'attend à ce que la méthode utilisant une commande soit plus lente, car cela implique plus de travail, mais d'un facteur de combien? À vous de le trouver.

En résumé, vous devez implémenter les 3 fonctions suivantes du fichier `fileops.c`

- bumpvers_fonction() incrémentation du numéro de version
- bumpvers_commande() incrément par l'appel au programme `bumpvers`
- do_benchmark() pour mesurer le temps d'exécution moyen

Votre travail sera corrigé sur le système Hopper. Assurez-vous que les matricules des membres de l'équipe soient inscrits dans le fichier `equipe.txt`. Faire l'archive avec `make dist`, puis envoyer l'archive sur le serveur de correction. Votre note sera attribuée automatiquement. Vous pouvez soumettre votre travail plusieurs fois et la meilleure note sera conservée. D'autres détails et instructions pour l'utilisation de ce système seront fournis.

Barème de correction

- Implémentation correcte de la fonction `bumpvers_fonction`: 40 points
- Implémentation correcte de la fonction `bumpvers_commande`: 30 points
- Implémentation correcte de la fonction `do_benchmark`: 30 points
- Qualité du code (fuite mémoire, descripteurs non-fermés, gestion d'erreur, etc): pénalité jusqu'à 10 points
- Total sur 100 points

Bon travail!