Multi Core Unix wc

On se propose de réimplémenter l'utilitaire unix `wc` en multi-cœurs. `whatis` nous dit

```
wc (1)       - print newline, word, and byte counts for each file
```

En supposant que l'exécutable se nomme `mwc`, la commande

```
mwc -w FILE
```

doit renvoyer le nombre de mots du fichier `FILE`. Pour compter le nombre de lignes, on utilisera `mwc -l FILE`, et pour compter le nombre de caractères, `mwc -c FILE`. Le cas sans option doit agir de la même manière qu'avec l'option `-w`. On ne traite qu'une seule option à la fois, le programme n'utilise pas l'entrée standard.

Le comportement (et la sortie) de votre `mwc` doit donc être rigoureusement le même que celui du `wc` Unix à option identique (sauf le cas sans option).

Le principe est de diviser le fichiers et d'attribuer un morceau de fichier par fil d'exécution.

`http://www.lsv.fr/~hondet/resources/archos/mwc.tar.gz` contient un Makefile et une batterie de tests. Pour lancer les tests, `make tests`. Pour générer des fichiers de taille arbitraire (pour tester les performances),

```
< /dev/urandom tr -dc '\n\t [:alnum:]' | head -cN > FILE
```

où `N` est la taille du fichier et `FILE` est le fichier de sortie.

Remarques

- On utilisera des headers (fichiers en `.h`).
- On privilégiera les appels système[1]. On utilisera donc `open` plutôt que `fopen` (`printf` est autorisé).
- Il faut toujours s'attendre à un échec d'un appel système. Afficher l'erreur avec `perror` sera un traitement suffisant.
- Le manuel Unix est très utile lorsqu'on programme avec les librairies Unix, la section 2 est dédiée aux appels système, la section 3 concerne les appels de librairie (les pages relatives à `pthread` sont en section 3).
- On fera attention à ne pas avoir de fuites mémoire. Pour s'en assurer, on peut utiliser l'outil `valgrind` :

  ```
  valgrind ./mwc file.plain
  ```

Challenge L'archive contient le binaire de ma solution (qui utilise 4 cœurs). Comparez, et faites mieux.

1. En règle générale, il vaut mieux privilégier les appels de librairies, plus génériques et portables.

Modalités de rendu Vous rendrez une archive tar compressée nommée `Nom_Prénom.tar.gz` dont l'architecture sera :

```
Nom_Prénom
├── readme.pdf
├── Makefile
├── main.c
├── main.h
├── additional files.c
├── additional files.h
└── tests
    └── …
```

où `readme.pdf` contient des indications si vous jugez cela nécessaire. On notera que les fichiers sont présents directement à la racine de l'archive. Le `Makefile` doit avoir deux cibles

- `bin` qui permet de générer le binaire et
- `tests` permettant de tester l'exécutable créé.

Pour rappel, une cible s'écrit

```
cible : dépendances
recette
```

où la recette est la commande déclenchée par la cible. Les dépendances peuvent être vides.

La propreté du code influencera la note. Vous pouvez utiliser des embellificateurs de code comme

- `http ://astyle.sourceforge.net/` ou
- `http ://uncrustify.sourceforge.net/` ou encore
- `https ://clang.llvm.org/docs/ClangFormat.html`.

Vous trouverez une archive `http ://www.lsv.fr/~hondet/resources/archos/mwc.tar.gz` pour démarrer. Elle contient l'arborescence demandée. Pour toute question, n'hésitez pas à contacter la hotline : `gabriel.hondet@lsv.fr`.