HPC - Projet 1 - Commande WC (Word Count)

Introduction

Le but de ce projet est d'essayer d'optimiser la commande `wc` qui fait partie des outils du cœur de Linux. Cette commande permet de base d'obtenir les informations suivantes sur un ou plusieurs fichiers, dans l'ordre d'affichage :

- le nombre de retour à la ligne
- le nombre de mots (séparés par un espace)
- le nombre d'octets (pas forcément le nombre de caractères suite à l'introduction d'Unicode et les caractères codés sur plusieurs octets)

Exemple:

```
$ wc ideas.txt excerpt.txt
     40     149     947 ideas.txt
   2294   16638   97724 excerpt.txt
   2334   16787   98671 total
```

`wc` peut encore compter d'autres choses spécifier le retour voulu avec ces différentes options:

```
  -c, --bytes             afficher le nombre d'octets
  -m, --chars             afficher le nombre de caractères
  -l, --lines             afficher le nombre de sauts de lignes
      --files0-from=F     lire l'entrée depuis les fichiers indiqués par
                          des noms terminant par NUL dans le fichier F
  -L, --max-line-length   afficher la largeur maximale d'affichage
  -w, --words             afficher le nombre de mots
      --help              afficher l'aide et quitter
      --version           afficher des informations de version et quitter
```

Motivation

J'ai choisi d'essayer d'améliorer les performances d'une commande Linux, car je suis intéressé à mieux comprendre le fonctionnement de cet OS. Je l'ai installé au début du semestre et je souhaite apprendre à utiliser tout son potentiel afin d'être à l'aise et avoir de la facilité dans ma future vie professionnelle.

Matériel

L'ordinateur sur lequel a été fait ce projet dispose comme CPU d'un Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz et ses caches sont:

- L1 Data 32 KiB (x2)
- L1 Instruction 32 KiB (x2)
- L2 Unified 256 KiB (x2)
- L3 Unified 4096 KiB (x1)

Prérequis

Voici la liste des prérequis à avoir pour pouvoir installer et lancer le projet:

Autoconf https://www.gnu.org/software/autoconf/

```
sudo apt-get install autoconf
autoconf --version
 autoconf (GNU Autoconf) 2.69
 Copyright (C) 2012 Free Software Foundation, Inc.....
```

Automake https://www.gnu.org/software/automake/

```
sudo apt-get install automake
which automake
 /usr/local/bin/automake
```

Bison https://www.gnu.org/software/bison/

```
wget http://mirror.rit.edu/gnu/bison/bison-3.6.tar.gz
tar xvzf bison-3.6.tar.gz
cd bison-3.6
./configure
make
make install
```

Gettext https://www.gnu.org/software/gettext/

```
apt-get install gettext
```

Si `autopoint` est manquant:

```
apt-get install autopoint
```

Git https://git-scm.com/

Gperf https://www.gnu.org/software/gperf/

```
apt-get install gperf
```

Gzip https://www.gnu.org/software/gzip/

Perl https://www.cpan.org/

Rsync https://rsync.samba.org/

Tar https://www.gnu.org/software/tar/

Texinfo https://www.gnu.org/software/texinfo/

```
wget https://ftp.gnu.org/gnu/texinfo/texinfo-6.7.tar.gz
tar xvzf texinfo-6.7.tar.gz
cd texinfo-6.7
./configure
make
```

Installation

Base

Faisant partie des outils du cœur, il faut cloner l'ensemble des outils (http://git.savannah.gnu.org/cgit/coreutils.git). La version avec laquelle a été fait ce projet est la *8.32*.

```
git clone git://git.savannah.gnu.org/coreutils.git
cd coreutils
git checkout v8.32
```

Le fichier *README-hacking* explique la marche à suivre pour pouvoir exécuter le dépôt. Il est nécessaire d'avoir la gnulib pour effectuer l'installation et taper `export GNULIB_SRCDIR=/path/to/gnulib` si la librairie n'a jamais été installer. Ici, elle est installée via le script `./bootstrap` dans le dossier `gnulib` de coreutils mais la variable GNULIB_SRCDIR doit être initialisée.

Il faut en premier lieu installer les prérequis qui se trouvent dans *README-prereq* (ceux cités plus haut). Une fois fait, le script `./bootstrap` peut être lancé ainsi que les commandes qui suivent:

```
./bootstrap
./configure --quiet #[--enable-gcc-warnings] [*]
make
```

Si `make` retourne l'erreur **lib/acl-internal.c:479:1: error: function might be candidate for attribute 'const' [-Werror=suggest-attribute=const] free_permission_context (struct permission_context *ctx)**, il faut rajouter `__attribute__((const))` à la déclaration de la fonction `free_permission_context` afin de supprimer le warning.

Normalement, il suffit de refaire un `make` lors du clonage du repot, car l'installation est déjà faite (exceptés les prérequis).

Stratégie de benchmark

Pour effectuer les tests, trois fichiers texte ont été créés avec des paragraphes aléatoires tirés de https://fr.lipsum.com/ . Ces fichiers ne sont pas dans le git car github n'accepte pas de fichiers plus grand que 100 Mo. Le tableau suivant résume leurs caractéristiques.

Fichier	# de Retour à la ligne	# de Mots	# d'Octets	# de Caractères
text.txt	35536896	1463205888	10727129088 (10.7 Go)	10727129088
text2.txt	2221056	91450368	670445568 (670.4 Mo)	670445568

Fichier	# de Retour à la ligne	# de Mots	# d'Octets	# de Caractères
text3.txt	1252814	51583724	378173149 (378.2 Mo)	378173149

Afin qu'il aie une mesure qui soit faite, il a fallu prendre des fichiers disposant de beaucoup de caractères. C'est pourquoi leur taille sont très grandes.

Pour chaque benchmark, la stratégie est de mesurer le temps d'exécution avec la commande `time` :

- le fichier *text.txt* exécuté seul
- les fichiers *text2.txt* et *text3.txt* exécutés ensemble

Il n'est pas nécessaire de tester les différentes options, car, comme le montrer l'image ci-dessous, elles n'ont pas d'effets sur le temps d'exécution. Cela n'est pas surprenant, avec ou sans option, il faut parcourir tout le fichier pour compter les mots, caractères, ... Tous les tests sont donc faits sans option.

```
(base) oem@PC-Cours:~/Documents/2019-2020/HPC/Projet/1 - WC/coreutils/src$ time ./wc -m ../../text.txt
10727129088 ../../text.txt

real    1m29.545s
user    1m13.396s
sys     0m5.650s
(base) oem@PC-Cours:~/Documents/2019-2020/HPC/Projet/1 - WC/coreutils/src$ time ./wc ../../text.txt
   35536896  1463205888 10727129088 ../../text.txt

real    1m28.002s
user    1m13.364s
sys     0m5.430s
```

Mesure de base

text.txt

Temps total	Temps user	Temps sys
1m24.727s	1m12.807s	0m5.093s

text2.txt + text3.txt

Temps total	Temps user	Temps sys
0m7.582s	0m7.037s	0m0.452s

Outils

Perf a été utilisé, avec ses options `cpu-clock` et `faults`, afin de trouver les endroits qui méritent une optimisation.

Analyse

Après avoir utilisé *perf* sur le fichier text3.txt, la première observation faite est que les fautes de page ne sont pas nombreuses (entre 5 et 10 pour près de 400Mo de texte) et elles sont émises par des fonctions extérieures à `wc`, donc inatteignables.

Pour les coups d'horloge, les 85% se trouvent dans dans la fonction `wc`. Un goulot d'étranglement se trouve à la ligne 482 `linepos++;`. Il peut être amélioré en `++linepos;`, car cela va seulement incrémenter la valeur de `linepos` et l'évalue sans devoir la stocker dans une variable temporaire.

Les CFLAGS du Makefile n'ont que `-O2` pour les optimisations. En changeant en `-O3`, il se peut qu'il aie un gain de temps.

Une autre possibilité d'amélioration serait d'agrandir la taille du buffer. Ceci peut poser des problèmes au niveau de la mémoire si il est trop grand.

Résultats

En changeant le flag de compilation de `-O2` en `-O3`, il y a eu un petit gain de performance:

Fichier	Temps total	Temps user	Temps sys
text.txt	1m19.520s	1m5.492s	0m5.003s
text2.txt + text3.txt	0m6.235s	0m5.921s	0m0.308s

Le fait de changer `linepos++;` en `++linepos;` n'apporte pas de gain de temps significatif. Les temps obtenus sont quasi identiques que ceux de base. Pareil lorsque la commande est exécutée avec un buffer plus grand.

Conculsion

La commande `wc` a déjà eu pas mal d'optimisations qui ont été faites. Par exemple: si la machine, où la commande est exécutée, considère qu'un caractère est équivalent à un byte, le comptage va se faire par byte et non par caractère. Un autre exemple est lors du comptage des lignes. Si une ligne est courte (juste un `\n` par exemple), il n'aura pas d'appel à `memchr` pour chercher le prochain retour à la ligne. Ces différentes optimisations ont rendu la tâche compliqué pour en trouver de nouvelles. De plus, le fait qu'un caractère puisse être encodé sur plusieurs bytes augmente la complexité du code assez compliqué à comprendre.

Pour ces différentes raisons, je n'ai pas pu trouver d'optimisation concrète