Exercice 2

Le chiffre de César est sans doute le système cryptographique le plus ancien... et le plus facile à décrypter. Il consiste à décaler les lettres d'un message. Ainsi, avec un décalage de 2, par exemple, A devient C, B devient D, et ainsi de suite jusqu'à Y qui devient A et Z qui devient B. Le décalage peut également être négatif : avec un décalage de -1, A devient Z, B devient A, C devient B, etc. Le décalage doit être compris entre -25 et +25 et le chiffrement doit s'appliquer aussi bien sur les minuscules que sur les majuscules, mais on ne tiendra pas compte des caractères accentués (non-ASCII).

On souhaite profiter de cet exercice pour comparer l'efficacité des primitives systèmes de gestion des fichiers et des fonctions de la bibliothèque standard d'entrées-sorties (fonctions travaillant sur des `FILE*`) puisque ces deux familles ont des fonctionnalités comparables. Pour cela on propose de rédiger un programme `cesar` ayant deux comportements distincts suivant le nombre d'arguments qui lui sont fournis :

- `cesar` *`decalage`* : avec un seul argument, votre programme doit utiliser `getchar` pour lire des caractères sur l'entrée standard, appliquer le chiffre de César sur les lettres (les caractères qui ne sont pas des lettres ne doivent pas être transformés) et envoyer le résultat sur la sortie standard avec `putchar`.
- `cesar` *`decalage n`* : avec deux arguments, votre programme doit utiliser utiliser les primitives systèmes `read` et `write` pour lire sur l'entrée standard et écrire sur la sortie standard. Vous procéderez par des lectures/écritures de blocs de n octets.

Voici des exemples de tests que vous devez réaliser :

- `echo Il Fait Beau > toto.txt ; ./cesar 1 < toto.txt`
  Affiche « `Jm Gbju Cfbv` » sur la sortie standard.
- `./cesar 1 < /usr/include/stdio.h`
  Affiche le fichier indiqué avec chiffrement résultant d'un décalage de 1.
- `./cesar 0 < /usr/include/stdio.h > toto.txt ; cmp /usr/include/stdio.h`
  N'affiche rien, car le fichier `toto.txt` est identique à l'original puisque le décalage est nul.
- `./cesar -2 < toto.txt | ./cesar 2 > titi.txt ; cmp toto.txt titi.txt`
  N'affiche rien, car les deux décalages inverses restituent un fichier `titi.txt` identique à l'original.
- `./cesar -2 < /bin/ls | ./cesar 2 > titi ; cmp /bin/ls titi`
  N'affiche rien (mêmes raisons que précédemment), même sur un fichier contenant des données binaires.
- `./cesar 3 1 < /usr/bin/doxygen > /dev/null`
  Très lent (environ 30 secondes sur la machine turing) car entrées/sorties octet par octet.

Ensuite :

1. Vous utiliserez la commande `time` (voire `/usr/bin/time`) pour déterminer le temps d'exécution de votre programme. Par exemple :

   ```
   /usr/bin/time ./cesar -1 2 < /usr/bin/doxygen > /dev/null
   ```

2. Prenez un fichier de taille conséquente (si vous n'en trouvez pas de plus gros, `/usr/bin/doxygen` fait 16 Mo sur turing) de telle sorte que les temps mesurés soient significatifs. Pour éliminez le ralentissement dû à l'affichage sur votre écran, redirigez la sortie standard de votre programme vers `/dev/null`.
3. Pour avoir des valeurs fiables, répétez chaque mesure de temps plusieurs fois (par exemple 5) et prenez la moyenne des temps utilisateur+système.
4. Déterminez le temps de la version avec les fonctions de bibliothèque, puis les temps de la version avec les primitives systèmes pour différentes valeurs de n (prenez des puissances de 2 : 2^0, 2^1, 2^2, etc. jusqu'à ce que vous obteniez des valeurs qui n'évoluent plus beaucoup).
5. Représentez graphiquement les mesures effectuées. Mentionnez les différents paramètres utilisés (taille du fichier, nombre de mesures), les valeurs mesurées et toutes autres informations ou commentaires utiles. Pouvez-vous expliquer le comportement constaté ?

Déposez sur Moodle une archive au format TAR contenant 2 fichiers : le source de votre programme et le document que vous avez rédigé au format PDF.