TP Système n°2

Question 1

La commande Unix `wc` permet de compter les lignes, mots et caractères d'un fichier texte :

```
$ wc /etc/passwd
21 59 1165 /etc/passwd
```

Écrire un programme `wcp` permettant de compter les lignes, mots et caractères de plusieurs fichiers. Le programme qu'il vous est demandé d'écrire doit :

- lancer un processus fils pour chacun des noms de fichiers passés en paramètre ;
- chaque fils doit exécuter le programme `wc` sur le fichier dont il s'occupe ;
- le père doit finalement afficher le nombre de fils qui ont échoué (par exemple parce que le fichier n'existe pas ou n'est pas lisible).

Le programme doit évidemment vérifier que sa syntaxe d'appel est correcte.

Exemple de sortie :

```
$ python3 wcp.py /etc/passwd /etc/group /etc/service
21 59 1165 /etc/passwd
26 26 368 /etc/group
1 échec(s)

$ python3 wcp.py /etc/passwd /etc/group
26 26 368 /etc/group
21 59 1165 /etc/passwd
```

Question 2

Écrire la commande `wcpsort` qui réaliserait la même chose que la commande `wcp` suivie d'un `sort -k x` (où *x* vaut 1, 2 ou 3, selon la position du champ à trier : voir man sort). La position du champ à trier sera fournie comme premier paramètre. Le premier champ a la position 1 (vous pouvez envisager que, si cette position n'est pas fournie, la position 1 sera prise par défaut).

Exemple :

```
$ python3 wcpsort.py 1 /etc/group /etc/passwd
24 36 1238 /etc/passwd
54 54 832 /etc/group
```

TIP Rien ne vous empêche (au contraire…) d'appeler la commande `wcp` à partir de `wcpsort`…

Question 3

On souhaite réaliser un programme mettant en jeu 2 processus (il s'agit d'un seul programme, pas de 2 programmes différents) :

- Un processus « client » qui gére l'interaction avec l'utilisateur (saisie au clavier des commandes et affichage du résultat).
- Un processus « serveur » qui traite la demande de l'utilisateur et produit une réponse.

Concrètement, lorsque le programme démarre, le processus client attend que l'utilisateur entre un nom de fichier et transmet ce nom au processus serveur.

- Si le fichier indiqué existe et est accessible en lecture, le serveur renvoie la réponse (donc le contenu du fichier) au processus client. Sinon, il envoie un message d'erreur au client.
- Le processus client se charge ensuite d'afficher cette réponse à l'écran puis attend un nouveau nom de fichier.
- Le programme se termine lorsque l'utilisateur tape la commande QUIT.

[TIP] Il est conseillé d'encapsuler le traitement du processus serveur et du processus client dans deux fonctions distinctes.

Amélioration possible

On veut maintenant pouvoir passer au programme un paramètre pour indiquer un répertoire de lecture, auquel cas les fichiers ne pourront être consultés que dans celui-ci. Si ce paramètre n'est pas fourni, on considère que le répertoire de lecture est le répertoire courant.

Question 4

On veut résoudre le même problème que celui de la question précédente, mais ici, on utilisera deux programmes distincts : `client.py` et `serveur.py`. Le premier jouera le rôle du client, le second jouera le rôle du serveur.

Plusieurs programmes clients pourront se connecter au serveur.