Exercice 1 (Compléments sur la manipulation de fichiers en ligne de commande: Notion de redirection)[1].

Un processus communique avec l'extérieur par l'intermédiaire de trois fichiers (appelés fichiers standard):

- Le fichier *entréé standard* sur lequel le processus lit les données.
- Le fichier *sortie standard* sur lequel le processus écrit ses résultats.
- Le fichier *sortie erreur* sur lequel le processus écrit ses messages d'erreur.

Chaque fois qu'un processus naît, ces trois fichiers sont ouverts;par défaut, ils sont associés au terminal (l'entrée standard est associée au clavier,la sortie standard et la sortie erreur standard sont associées à l'écran).

Il est possible de *rediriger* les entrés/sorties standard d'un processus , c'est-à-dire e leur associer un fichier autre que le terminal.

On utilise la syntaxe suivante:

- *commande* < *chemin d'un fichier* .
  Redirige l'entrée standard de *commande* sur le fichier dont on donne le chemin
- *commande* > *chemin de fichier*: redirige la sortie standard avec *écrasement* du fichier désigné par chemin.
- *commande* >> *chemin de fichier*: redirige la sortie standard *sans écrasement* du fichier désigné par chemin.
- *commande* 2> *chemin de fichier* : redirige la sortie erreur standard *avec écrasement* du fichier désigné par chemen.
- *commande* 2>> *chemin de fichier* : redirige la sortie erreur standard *sans écrasement* du fichier désigné par chemin.

Tester les instructions suivantes écrites dans l'invite de commande du shell:

- ls -l
- ls -l > fichiers.txt
- wc /etc/rpc
- wc /etc/rpc > composants
- ls rrrrr
- ls rrrrr 2> erreurs
- gedit erreurs

[1] D'après "Passeport pour Unix et C" (JM.Champarnaud et G.Hansel-Edition Vuibert)

Exercice 2 (les commandes **touch,cat,wc**)

Dans cet exercice on travaille dans un *Terminal* sous linux ubuntu et on utilise les commandes du *shell*.
Pour chacune des questions suivantes , testez les commandes proposées et/ou exécutez la tâche demandée.

1. Dans le *Terminal* , créér un dossier nommé *Sys-Exploitation* puis un sous-dossier nommé *Villes*.
   Dans la suite de l'exercice on se place dans le dossier *Sys-Exploitation*.
2. Créér un fichier nommé *toulouse.txt* avec la commande *touch*.

   ```
   touch toulouse.txt
   ```

   Pour le moment le fichier est vide.
3. Copier le fichier *toulouse.txt* dans le dossier *Villes* en utilisant la commande **cp**.
4. On va écrire dans le fichier *toulouse.txt* en utilisant la commande *cat*.

   ```
   cat > Villes/toulouse.txt

   Place du Capitole,Toulouse
   Couvent des Jacobins,Toulouse
   ```

   (Pour arrêter faire Ctrl+D)
5. Afficher alors le contenu du fichier *toulouse.txt* à l'aide de la commande *cat*.
   Pour continuer à écrire dans le fichier *toulouse.txt* sans écraser le précédent contenu , on écrit :

   ```
   cat >> Villes/toulouse.txt

   La basilique Saint-Sernin,Toulouse
   Lycée Pierre de Fermat,Toulouse
   ```
6. Créer dans le dossier *Villes* un fichier texte 'bordeaux.txt' dans lequel vous inscrirez les lignes:
   Place de la Bourse,Bordeaux
   Rue Mazarin,Bordeaux
   Place Montaigne,Bordeaux
7. On peut concaténer le contenu de deux fichiers 'toulouse.txt' et 'bordeaux.txt' et copier le résultat dans un nouveau fichier 'villes-sud-ouest':

   ```
   cat 'toulouse.txt' 'bordeaux.txt' > 'villes-sud-ouest'
   ```
8. La commande *wc* affiche le nombre de lignes , de mots et de caractères d'un fichier donné en argument:

   ```
   wc /Villes/toulouse.txt
   ```

   Cette commande peut être utilisée avec les options -l (n'affiche que le nombre de lignes), -w (n'affiche que le nombre de mots) et -c (n'affiche que le nombre de caractères).

Exercice 3 (la commande **grep)**

la commande *grep* sélectionne sur son entrée standard les lignes contenant une chaîne de caractères obéissant à un modèle défini par une expression.
Les lignes sélectionnées sont écrites sur la sortie standard.

1. Exemple:

   ```
   grep 'Place' Villes/toulouse.txt
   ```

   Résultat: Affiche toutes les lignes du fichier 'toulouse.txt' contenant la chaîne 'Place'.

   Afficher alors toutes les lignes des fichiers .txt du répertoire *Villes* et qui contiennent la chaîne 'Place'.

   Remarque : On utilise l'option -l pour afficher seulement les noms des fichiers contenant la chaîne 'Place' .

   ```
   grep -l 'Place' Villes/*
   ```

2. Copier toutes les lignes du fichier Villes/bordeaux.txt contenant la chaîne 'Place' dans un fichier Villes/nouveau.txt (utilisez la redirection >)

Exercice 4 (tubes)

On peut lancer deux processus de façon simultanée qui échangent des données grâce à une zone de mémoire gérée par le système, appelée *pipe* (anglais) ou *tube* en français:

```
commande 1 | commande 2
```

Le système crée un tube et deux processus

- *commande 1* lance un premier processus dont la sortie standard est redirigée vers le tube.
- *commande 2* lance un second processus dont l'entrée standard est redirigée vers le tube.

Les deux processus partagent l'accès au processeur.C'est le système qui s'occupe de leur synchronisation:
Le processus lecteur (commande 2) est mis en sommeil tant que le tube est vide;le processus écrivant (commande 1) est mis en sommeil si le tube est plein.
On peut se représenter un *tube* comme une file (fifo) : les caractères sont lus dans l'ordre où ils ont été écrit[2].

1. Tapez dans Terminal , la ligne:

   ```
   grep 'Place' Villes/*.txt | wc -l
   ```

   Que fait cette commande?

2. Ecrire à l'invite de commande une instruction qui copie dans un fichier *'nombre.txt'* toutes les lignes des fichiers .txt du dossier *Villes* qui contiennent la chaîne 'Place'.

3. Dans un dossier , on trouve des fichiers avec des extensions variées : .txt , .pdf , .py etc.
   Utiliser un tube | pour afficher seulement les fichiers avec la chaîne 'py'.
   Expliquer la différence avec la commande ls *.py

4. Faire de même pour afficher le nombre de fichiers ayant pour extension 'py'

[2] D'après "Passeport pour Unix et C" (JM.Champarnaud et G.Hansel-Edition Vuibert)

Exercice 5

1. Ouvrir le Terminal.
   Tapez *firefox* & dans le shell (l'option & permet à un programme de s'exécuter en arrière plan , le shell restant disponible pour de nouvelles commandes).
   Le **PID** du processus en cours s'affiche.Par exemple:
   [2] 10767
2. Lancer dans le terminal le programme gedit en arrière plan.
3. Taper la commande **ps -l**.

Exercice 6
On donne le programme *process-00.py*:

```
from multiprocessing import Process
from os import getppid


def recupere_pid():
    # on récupère le pid du processus parent
    pid = getppid()
    # on affiche
    print(pid)


if __name__ == '__main__':
    # Création d'un processus
    enfant = Process(target=recupere_pid)
    # On démarre le processus enfant
    enfant.start()
    # On attend que le processus enfant se termine
    enfant.join()
```

1. Le programme précédent montre comment on peut récupérer le *pid* d'un processus parent.
   Ouvrir le Terminal puis exécutez le programme précédent dans l'invite de commande.
2. Exécuter la commande ps -l & python3 process-00.py puis comparer avec le résultat obtenu précédemment.
3. Modifier le programme précédent pour obtenir l'affichage du *pid* du processus enfant (on utilisera enfant.pid qui a pour valeur le pid du processus enfant) , puis exécuter à nouveau la commande de la question 2)
4. Interprétez les résultats affichés.Quel est alors le pid du parent du programme ayant exécuté `process-00.py`?

Exercice 7

1. ```
   from time import sleep
   from multiprocessing import Process



   def fonction_f():
   ```
   ```

```
    sleep(3)


if __name__ == '__main__':

    enfant = Process(target=fonction_f)
    # On démarre le processus enfant
    enfant.start()
    #on teste si le processus créé est vivant
    print(enfant.is_alive())
    # On attend que le processus enfant se termine
    enfant.join()
    #on refait le test
    print(enfant.is_alive())
```

2. Comment peut-on interpréter les résultats affichés par ce programme?Expliquer l'instruction enfant.is_alive().