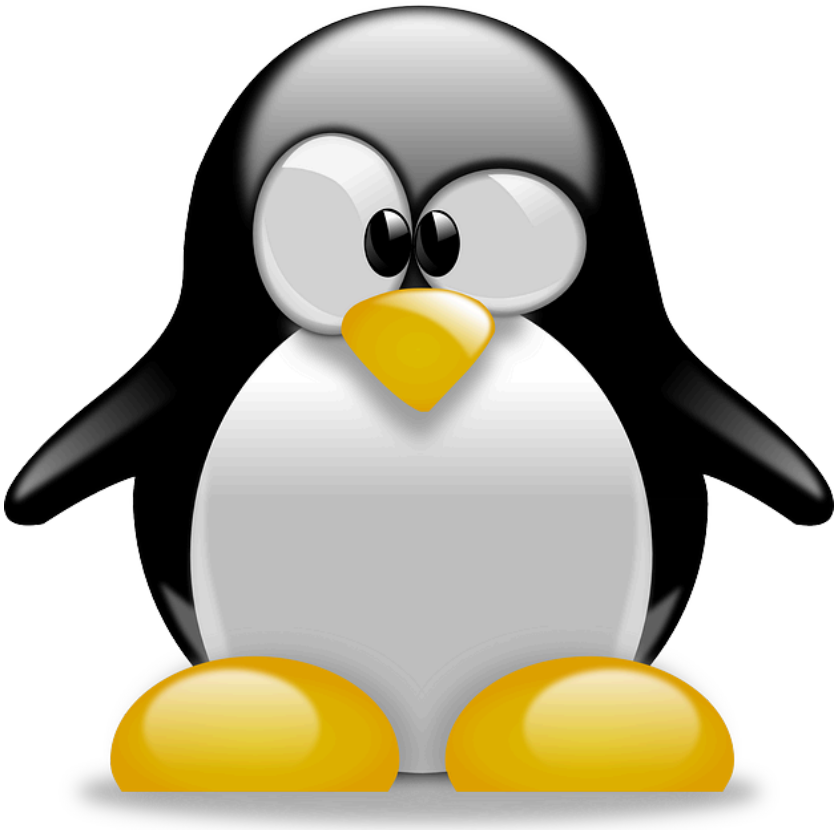
Fusi Belaid
Thomas
4A

Projet 2
ProgSys



1. Introduction

Ce TP de programmation système a pour objectif de reproduire, en Python, le fonctionnement de commandes Unix avec redirections, en utilisant des fonctions bas niveau. La première partie consiste à implémenter la commande `cat > fichier`, qui permet d'écrire des entrées clavier dans un fichier, tandis que la deuxième partie consiste à reproduire la commande `ls -l | grep \.py`, qui filtre les fichiers `.py` à l'aide d'un tube de communication (`pipe`). Ce projet met en pratique des concepts avancés comme la gestion des processus, les pipes, et les redirections standard avec `dup2`. L'exercice offre une compréhension approfondie du fonctionnement des commandes Unix et des mécanismes de communication inter-processus.

2. La commande cat > fichier

L'objectif du projet est d'écrire un script python bas niveau permettant de rentrer du texte dans un document à la manière de la commande cat > fichier.

Voici donc le code en question:

```
import os
import sys

if len(sys.argv) != 2:
   os.write(2, "Usage: python3 script.py
<nom_fichier>\n".encode()) #sortie 2 pour les erreurs
   os._exit(1)

nom_fichier = sys.argv[1]
try:
   fd = os.open(nom_fichier, os.O_WRONLY | os.O_CREAT |
os.O_TRUNC, 0o644)

except OSError as e:
   os.write(2,"Erreur lors de l'ouverture du fichier:
{e.strerror}".encode())
   os._exit(1)
os.write(1,"Entrez le texte à écrire puis tapez Ctrl-D pour
terminer:\n".encode())

try:
   while True:
       data = os.read(0, 1024)
       if not data:
           break
       os.write(fd, data)

   os.close(fd)
   os._exit(0)
except OSError as e:
   os.write(2, "Erreur de lecture".encode())
```

Et voici les tests:

```
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$ python3 script.py test.txt
Entrez le texte à écrire puis tapez Ctrl-D pour terminer:
aaa
bbb
ccc
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$ cat test.txt
aaa
bbb
ccc
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$
```

Ici on teste d'écrire des lettres et des sauts de ligne dans un fichier test.txt qui existe déjà.

```
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$ ls
script.py  test.txt
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$ python3 script.py test2.txt
Entrez le texte à écrire puis tapez Ctrl-D pour terminer:
coucou
salut
bonjour
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$ cat test2.txt
coucou
salut
bonjour
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$
```

Ici l'on teste avec un fichier qui n'existe pas et qui est donc créé.

```
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$ python3 script.py test.txt
Entrez le texte à écrire puis tapez Ctrl-D pour terminer:
test@#!
$£µ*&§:
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$ cat test.txt
test@#!
$£µ*&§:
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet2$
```

On teste avec des caractères spéciaux et on voit que ça fonctionne.

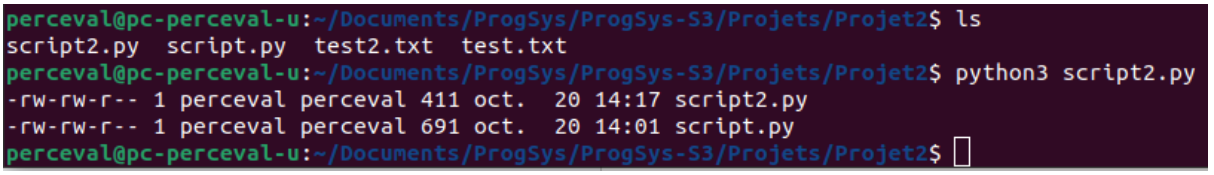
3. commande ls et grep par un tube

L'objectif de ce projet est de créer un programme permettant d'effectuer l'équivalent de la commande "ls -l | grep \\.py" avec des appels python de bas niveaux.

Voici donc le code en question:

```
import os
import sys


rd, wd = os.pipe()

pid1 = os.fork()

if pid1 == 0: # 1er fils
   os.dup2(wd, 1)
   os.close(rd)
   os.close(wd)
   os.execvp("ls", ["ls", "-l"])
   sys.exit(1)

pid2 = os.fork()

if pid2 == 0: # 2e fils
   os.dup2(rd, 0)
   os.close(rd)
   os.close(wd)
   os.execvp("grep", ["grep", "\\.py"])
   sys.exit(1)



os.close(rd)
os.close(wd)

os.waitpid(pid1, 0)
os.waitpid(pid2, 0)
```

Et voici les tests:



On a donc bien 2 fichiers .py dans notre répertoire et c'est également le résultat du script python.

4. Conclusion

Ce TP nous a permis d'acquérir une meilleure compréhension des mécanismes sous-jacents aux redirections de flux dans les systèmes Unix, notamment par l'utilisation des fonctions bas niveau telles que `os.read`, `os.write`, et `dup2`. Grâce à l'implémentation des commandes `cat > fichier` et `ls -l | grep \.py`, nous avons pu manipuler les tubes de communication et les processus de manière précise. Ces compétences sont essentielles pour maîtriser la gestion des processus dans un environnement Unix/Linux, et elles constituent une base solide pour des applications plus complexes dans la gestion de systèmes.