TP2.2 (3h) : Communication entre processus par tubes

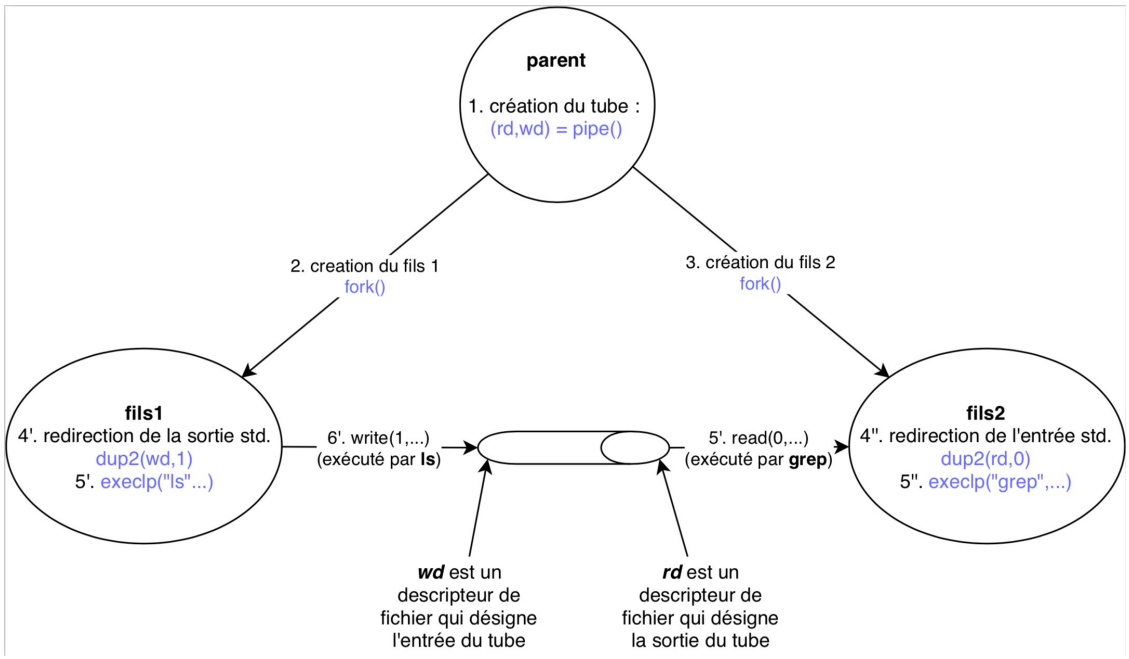
Objectif. *L'objectif de ce TP est de vous familiariser avec les fonctionnalités de base de l'API système POSIX pour la communication entre processus via des tubes*

Rappel sur la documentation. Les fonctions du module **os** en Python sont des copies conformes des fonctions système en langage C, qui disposent de pages manuel UNIX (**man**). La documentation du module Python **os** étant souvent assez succincte, il est utile de consulter également les pages **man** des fonctions C utilisées.

Ex. 1. Combiner pipe et dup pour créer un tube entre deux commandes

Écrire un programme qui crée un tube de communication (avec **pipe**) puis crée 2 processus fils, le premier exécutant la commande **ls -l**, le second exécutant la commande **grep \\.py**. Le tube doit préalablement être raccordé (avec **dup2**) de manière à reproduire l'équivalent de la commande shell **ls -l | grep \\.py**. Le processus père doit attendre que les deux fils se terminent avant de s'arrêter.

Le schéma suivant décrit la solution et les principaux pas exécutés par les trois processus. Ce schéma n'est cependant pas complet, notamment il faudra fermer les extrémités du tube inutilisées dans chacun des trois processus pour que tout fonctionne bien…



*A noter : **grep** continuera à tourner tant qu'il peut y avoir des données qui arrivent par le tube, c'est à dire tant que le côté **wd** reste ouvert dans l'un des processus. Pour que le programme s'arrête correctement, il faut veiller, dans chacun des processus à toujours fermer les descripteurs non-utilisés avec la fonction **close.***

```python
import os
import sys
import traceback

try:
    (fd1, fd2) = os.pipe()
    pid1 = os.fork()
    if pid1 == 0:
        os.close(fd1)
        os.dup2(fd2, 1)
        os.execlp("ls", "ls")
    else:
        pid2 = os.fork()
        if pid2 == 0:
            os.close(fd2)
            os.dup2(fd1, 0)
            os.execlp("grep", "grep", "\\.py")
        else:
            os.wait()
except OSError as e:
    traceback.print_exc()
    print(e.strerror)
    exit(1)
```

You, seconds ago • Uncommitted changes

Ex. 2. Communication par tube nommé

*Un tube nommé permet à deux processus quelconques (sans lien de parenté) de communiquer des données. Il est créé avec la **commande shell mkfifo** et il peut ensuite être utilisé comme un fichier ordinaire (il apparaît dans l'arborescence des fichiers, on l'ouvre avec **open**,...), avec la différence que les données ne sont pas stockées sur le disque, qu'un **read** effectué dessus est bloquant tant qu'il n'y a pas de données envoyées dans le tube, et que les données disparaissent du tube après la lecture.*

Essayer la séquence de commandes shell suivante:

```
mkfifo tmp
ls -l tmp
cat tmp
```

Dans un autre terminal, exécuter la commande "`cat > tmp`" et saisir quelques lignes de texte. Regarder le premier terminal et expliquer le résultat (en faisant appel à "**man mkfifo**").

Écrire **deux programmes** qui communiquent par le tube "tmp" créé précédemment. Le premier programme doit écrire dans le tube, en boucle toutes les secondes, des nombres aléatoires entre 32 et 99 (l'encodage sur 1 octet des

nombres envoyés se fait avec les fonctions vues en TD2, Ex.4). Le deuxième programme doit récupérer les nombres et écrire sur la sortie standard les caractères ASCII correspondants aux codes numériques reçus.

Tester le comportement du système si l'un ou l'autre des deux programmes est interrompu par CTRL-C. Que peut-on en déduire ?

P1 :

Exo1.py ● Exo2_1.py ×

S3 > PSE > TP2.2 > Exo2_1.py > ...

You, a week ago | 1 author (You)

```python
import os
import random
import traceback
import time
import sys

try:
    fd = os.open("tmp", os.O_WRONLY)
    while True:
        n = random.randint(32, 99)      You, a week ago
        os.write(fd, n.to_bytes(4, sys.byteorder))
        time.sleep(1)
except OSError as e:
    traceback.print_exc()
    print(e.strerror)
    exit(1)
```

P2 :

Exo1.py ● Exo2_2.py ×

S3 > PSE > TP2.2 > Exo2_2.py > ...

You, a week ago | 1 author (You)

```python
import os
import random
import traceback
import time

try:
    fd = os.open("tmp", os.O_RDONLY)
    while True:      You, a week ago •
        os.write(1, os.read(fd, 4))
except OSError as e:
    traceback.print_exc()
    print(e.strerror)
    exit(1)
```

Ex. 3. Communication bidirectionnelle et tubes

On souhaite implémenter un programme permettant de réaliser le comportement suivant :

- il lance deux processus s'échangeant des données à l'aide de tubes anonymes
- la communication entre les deux processus doit être bidirectionnelle
- le processus P1 doit lire le contenu d'un fichier ff.txt et le communiquer au processus P2
- le processus P2 doit afficher à la sortie d'erreurs le contenu reçu (en provenance de P1)
- le processus P2 doit lire le contenu d'un fichier gg.txt et le communiquer au processus P1
- le processus P1 doit afficher à la sortie standard le contenu reçu (en provenance de P2)

```
You, seconds ago | 1 author (You)
import os
import traceback

try:
    fd1, fd2 = os.pipe()
    fd3, fd4 = os.pipe()
    pid1 = os.fork()
    if pid1 == 0:
        fdF = os.open("ff.txt", os.O_RDONLY)
        buffer = os.read(fdF, 256)
        while len(buffer) > 0:
            os.write(fd2, buffer)
            buffer = os.read(fdF, 256)
        os.write(1, os.read(fd3, 256))
    else:
        pid2 = os.fork()
        if pid2 == 0:
            os.write(2, os.read(fd1, 256))
            fdG = os.open("gg.txt", os.O_RDONLY)
            buffer = os.read(fdG, 256)
            while len(buffer) > 0:
                os.write(fd4, buffer)
                buffer = os.read(fdG, 256)
        else:
            os.wait()
except OSError as e:
    traceback.print_exc()
    print(e.strerror)
    exit(1)
        You, seconds ago • Uncommitted changes
```

POUR ALLER PLUS LOIN (exercice optionnel) :

Ex. 4. Communication bidirectionnelle par sockets

Il existe un moyen plus simple de mettre en place une communication bidirectionnelle entre processus, sans pour autant passer par toute la complexité de la création d'une connexion réseau. L'appel ***socket.socketpair*** *permet de créer une paire de sockets connectés, chacun pouvant être utilisé en lecture/écriture. Chaque socket dispos d'un descripteur bas-niveau, accessible avec la méthode* ***fileno****, qui peut être utilisé avec* ***read, write, close****, etc.*

La paire de sockets peut utiliser différents protocoles pour communiquer, pour cet exercice vous pouvez utiliser l'appel suivant pour la création : ***socket.socketpair(socket.AF_UNIX, socket.SOCK_STREAM)****.*

Reprendre le code de l'exercice 3 et le transformer pour utiliser une paire de sockets à la place des tubes. Pour simplifier, l'envoi du contenu des deux fichiers par les deux processus fils peut se faire en même temps (avant qu'ils commencent à lire les données arrivant par le socket).

```python
import os
import traceback
import socket

try:
    #fd1 : Ecriture du contenue de ff.txt dans le premier fils et lecture dans le second
    #fd2 : Ecriture du contenue de gg.txt dans le second fils et lecture dans le premier
    fd1, fd2 = socket.socketpair(socket.AF_UNIX, socket.SOCK_STREAM)
    pid1 = os.fork()
    if pid1 == 0:
        #fdF : Fichier ff.txt
        fdF = os.open("ff.txt", os.O_RDONLY)
        buffer = os.read(fdF, 256)
        while len(buffer) > 0:
            os.write(fd1.fileno(), buffer)
            buffer = os.read(fdF, 256)
        os.write(1, os.read(fd2.fileno(), 256))
    else:
        pid2 = os.fork()
        if pid2 == 0:
            #fdG : Fichier gg.txt
            fdG = os.open("gg.txt", os.O_RDONLY)
            buffer = os.read(fdG, 256)
            while len(buffer) > 0:
                os.write(fd2.fileno(), buffer)
                buffer = os.read(fdG, 256)
            os.write(2, os.read(fd1.fileno(), 256)) #Fonctionn si placé ici. Ne fonctionne pas si placé avant l'ouverture de gg.txt
        else:
            os.wait()
except OSError as e:
    traceback.print_exc()
    print(e.strerror)
    exit(1)
    You, seconds ago • Uncommitted changes
```