Université Côte d'Azur
Faculté des Sciences - Département d'Informatique

Licence d'Informatique L2
Introduction aux Systèmes et Réseaux

TP n°4 : Fichiers, tubes, et entrées-sorties

1 Écrivains simultanés

Lire le programme suivant, et prédire ce qu'il doit afficher.

```
import os, sys

pid = os.fork()
if pid == 0:
    for i in range(300):
        os.write(1, b'a')
    sys.exit(0)
for i in range(300):
        os.write(1, b'b')
sys.exit(0)
```

L'exécuter plusieurs fois et vérifier que la prédiction est correcte.

2 Manipulation de tubes

Pour permettre la communication entre un père et son fils, il est possible d'utiliser un tube créé par la primitive os.pipe(). Cette primitive retourne comme résultat un couple d'entiers, qui sont les descripteurs des fichiers correspondant à l'entrée et à la sortie du tube. Si la création échoue, la primitive lève une exception ; sinon, le tube est créé en mémoire, l'élément 1 du tableau contient le numéro de l'entrée, et l'élément 0 le numéro de la sortie. On peut alors écrire et lire dans le tube au moyen des primitives `os.read()` et `os.write()`, et chaque processus peut fermer sa partie inutilisée du tube à l'aide de `os.close()`.

1. Écrire un programme qui crée un tube, puis un fils (rappelons que le fils hérite des descripteurs du père). Le père devra fermer la sortie du tube, puis écrire tout ce qu'il lit au clavier (descripteur 0) dans le tube. Le fils devra fermer l'entrée du tube, puis écrire tout ce qu'il lit dans le tube vers l'écran (descripteur 1).
2. Même question, mais cette fois le père crée un tube, puis deux fils. Le fils 1 envoie dans le tube ce qu'il lit au clavier, et le fils 2 affiche à l'écran ce qu'il lit dans le tube. Imaginer une méthode pour terminer proprement le déroulement de ces opérations. Note : Une lecture depuis un tube fermé à l'autre bout (car le processus écrivain s'est par exemple terminé) retournera un séquence d'octets de taille nulle, i.e. une

fin de fichier. Une écriture dans un tube fermé à l'autre bout provoquera l'envoi par le noyau d'un signal `SIGPIPE` qui provoquera la levée d'une exception `broken pipe`. Il faut donc veiller à fermer les deux extrémités d'un tube dans le bon ordre.

3. La primitive `os.dup()` sert à dupliquer un descripteur de fichier en utilisant le plus petit numéro de descripteur non utilisé. Elle prend en argument un descripteur de fichier ouvert, et retourne le descripteur de la copie. Pour rediriger la sortie standard à l'écran (resp. l'entrée standard au clavier), l'idée est de fermer le descripteur 1 (resp. 0), puis de faire un `os.dup()` ou un `os.dup2()` sur le descripteur qui deviendra la nouvelle sortie standard (resp entrée standard). Dans les deux cas, il ne faudra pas oublier de fermer également le descripteur original (celui qu'on vient de dupliquer), puisqu'il ne sert alors plus à rien.
   Refaire la question 1 ci-dessus en redirigeant la sortie standard du père vers l'entrée du tube et l'entrée standard du fils vers la sortie du tube.
   Lancer ensuite dans chacun des processus la commande `/bin/cat` au moyen d'`os.execv()`.

3 Client-serveur avec tubes et FIFOS

3.1 Client-serveur avec tubes

Faire fonctionner les programmes `mainpipe.py`, `client.py`, `serveur.py` vus en TD.

3.2 Client-serveur avec FIFOs

1. Comme cela a été vu en TD, transposer les programmes ci-dessus pour les faire fonctionner avec des FIFOs au lieu de tubes simples.
2. Écrire le programme d'un client et d'un serveur indépendants (non liés par une relation père-fils) et communiquant par des FIFOs.

3.3 `tr` en version client-serveur

Programmer une version de la commande `tr` en version client-serveur. Ce programme nécessite 2 arguments en ligne de commande : des chaînes de caractères *utf-8 entree* et *sortie* de même longueur. Tout ce que le client envoie au serveur est retourné au client. Cependant, chaque caractère figurant dans *entree* sera remplacé par le caractère de même indice de *sortie*. Le travail de traduction est effectué du côté serveur. Une possibilité est d'utiliser la méthode `replace` des chaînes de caractères.

1. la première version grâce à `os.pipe()`.
2. la seconde version grâce à `os.mkfifo()`, les deux processus étant indépendants.

4 `mytime.py` amélioré

Rajoutez l'option `-no-output` à la commande `mytime.py` :

```
mytime.py [-n k] [-s] [--no-output] commande [arg ...]
```

Cette option supprime tout affichage de `commande`. Pour cela, il faut rediriger les sorties (standard et d'erreur) de `commande` vers le fichier `/dev/null` grâce à `os.dup` ou `os.dup2`.

Pour pouvoir, à moment donné, restaurer le comportement par défaut, il faut au préalable penser à sauvegarder les descripteurs de fichiers initiaux (là encore grâce à `os.dup` ou `os.dup2`).