Université de Nice Sophia Antipolis
UFR Sciences - Dépt. Informatique

Licence d'Informatique L2
Introduction aux Systèmes et Réseaux

(feuille d'exercice adaptée du Cours de S. Krakowiak, U. Grenoble)

TD n° 3 : Fichiers, tubes, et entrées-sorties

L'objectif de ce TD [1] est d'illustrer quelques notions relatives aux communications à travers les fichiers et les tubes dans Unix.

1 Descripteurs de fichiers

On rappelle qu'un descripteur de fichier est créé lors de l'ouverture d'un fichier par un processus. Un descripteur est identifié par un entier (numéro de descripteur), qui est une entrée dans la table des descripteurs du processus. Dans la suite, lorsqu'il n'y a pas d'ambiguïté, nous utilisons aussi le terme de descripteur pour désigner les numéros de descripteurs. Les trois descripteurs 0, 1, et 2 sont respectivement attribués, dans chaque processus, au flot d'entrée standard `stdin`, au flot de sortie standard `stdout` et au flot de sortie d'erreur `stderr`. Les descripteurs servent de paramètres aux primitives de manipulation de fichiers telles que `read()`, `write()`, etc.

1. Qu'affiche le programme ci-après ?

```
#! /usr/bin/env python
import sys,os

fd1 = os.open("toto.txt", os.O_RDONLY)
os.close(fd1)
fd2 = os.open("titi.txt", os.O_RDONLY)
print("fd2 = %d" %(fd2))
sys.exit(0)
```

2. On suppose que le fichier `toto.txt` contient dans cet ordre les 6 caractères ASCII "`lambda`". Qu'affiche le programme ci-après ?

```
#! /usr/bin/env python
import sys,os

fd1 = os.open("toto.txt", os.O_RDONLY)
fd2 = os.open("toto.txt", os.O_RDONLY)
c = os.read(fd1, 1)
c = os.read(fd2, 1)
print("c = %c"%(c))
sys.exit(0)
```

1. Certains exemples sont empruntés (avec des adaptations) aux ouvrages suivants : R. E. Bryant, D. O'Hallaron. *Computer Systems: a Programmer's Perspective*, Prentice Hall, 2003, et W. R. Stevens, *UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications*, Prentice Hall, 1999.

3. Rappelons qu'un processus fils hérite des descripteurs des fichiers ouverts de son père. Comme dans la question précédente, le fichier `toto.txt` contient les 6 caractères ASCII "`lambda`". Qu'affiche le programme ci-après ?

```
#! /usr/bin/env python
import sys,os

fd = os.open("toto.txt", os.O_RDONLY)
pid = os.fork()
if ( pid == 0):
    c = os.read(fd, 1)
    sys.exit(0)
os.wait()
c = os.read(fd, 1)
print("c = %c" %(c))
sys.exit(0)
```

4. 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. Cette primitive sert à rediriger un flot d'entrée ou de sortie.

   La primitive `os.dup2()` est une variante de `os.dup()`, qui prend deux arguments : le descripteur du fichier à dupliquer, et le nouveau descripteur (qui recevra la copie du premier). Si ce nouveau descripteur est occupé, il sera fermé au préalable.
   Le fichier `toto.txt` contient toujours les 6 caractères ASCII "`lambda`". Qu'affiche le programme ci-après ?

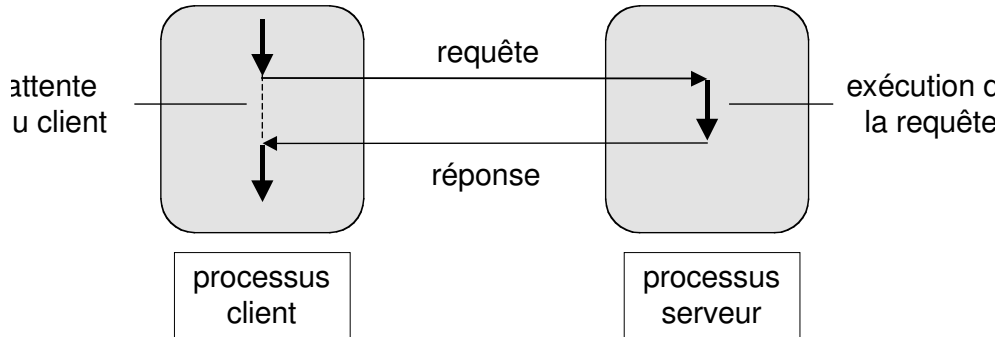
```
#! /usr/bin/env python
import sys,os

fd1 = os.open("toto.txt", os.O_RDONLY)
fd2 = os.open("toto.txt", os.O_RDONLY)
c = os.read(fd2, 1)
os.dup2(fd2, fd1)
c = os.read(fd1, 1)
print("c = %c" %(c))
sys.exit(0)
```

2 Manipulation de tubes

Rappels. Un tube (en anglais *pipe*) est un fichier particulier qui fonctionne en FIFO, et sert de canal de communication entre deux processus. Un tube possède un flot d'entrée et un flot de sortie, et les données sont lues sur le flot de sortie dans l'ordre selon lequel elles sont écrites dans le flot d'entrée. Un tube peut être crée au niveau du langage de commande par l'opérateur `|`, ou au niveau des appels systèmes par la primitive `os.pipe()`.

On a vu dans le cours comment un processus père peut communiquer vers son fils via un tube. Il est facile de créer un autre tube permettant la communication en sens inverse (du fils vers le père). On a alors tout ce qu'il faut pour construire un système simple fonctionnant en client-serveur.

Le schéma client-serveur est un schéma de communication entre deux processus, fonctionnant sur un mode synchrone requête-réponse : le processus client envoie une requête au processus serveur, et attend la réponse. Le processus serveur exécute la requête, et envoie la réponse au client. Quand ce dernier reçoit la réponse, il reprend son exécution.



Ce schéma sera vu en détail lors du cours sur les réseaux, et appliqué au cas où le client et le serveur sont sur deux machines différentes. Pour le moment, on se limite à des processus sur la même machine.

2.1 Tubes ordinaires

Un tube ordinaire n'a pas de nom et n'est accessible que par des descripteurs. Donc, en pratique, il ne peut être utilisé qu'entre ascendants et descendants (ou entre descendant du même processus). On va donc programmer le processus serveur comme un fils du client.

Le programme ci-après (fichier `mainpipe.py`, disponible sur le wiki) est un programme générique qui exécute un processus client et un serveur fils de ce client. Les programmes du client et du serveur doivent être fournis sous les noms `client.py` et `server.py`. Chacun d'eux prend deux paramètres, le descripteur du flot d'entrée et celui du flot de sortie, que le programme principal connecte à des tubes.

À titre d'exemple (les programmes sont également sur wiki), on écrit un serveur `server.py` qui renvoie au client le contenu d'un fichier dont le nom lui est transmis par le client. Le client `client.py` lit au clavier le nom du fichier et affiche le contenu à l'écran. Cet exemple illustre d'une part l'utilisation des tubes, d'autre part celui de l'ouverture et de la lecture d'un fichier.

On demande dans un premier temps d'examiner ces programmes et de bien comprendre leur fonctionnement. Noter que les opérations sur les tubes (`os.open`, `os.close`, `os.read`, `os.write`) sont les mêmes que celles sur les fichiers. Noter l'usage de quelques fonctions de la bibliothèque C pour la manipulation de chaînes de caractères. Ces programmes seront expérimentés au cours du TP.

```
#! /usr/bin/env python
import sys,os
# programme générique client-serveur

# Les deux modules suivants doivent etre ecrit
# ils sont supposes fournir les deux methodes
```

```
# client.client()
import client,server

if __name__ == "__main__":
   (rfd1,wdfd1) = os.pipe()            # tube pere vers fils
   (rfd2,wfd2) = os.pipe()            # tube pere vers fils
   childpid = os.fork()
   if (childpid == 0):
       os.close(wfd1)
       os.close(rfd2)
       server.server(rfd1, wfd2)      # fils execute serveur
       sys.exit(0)
   os.close(rfd1)
   os.close(wfd2)
   client.client(rfd2, wfd1)          # pere execute client
   os.waitpid(childpid, 0)            # attendre fin fils
   sys.exit(0)
```

```
#! /usr/bin/env python
# file: client.py (module client)

import sys,os

def client(readfd, writefd):
   global MAXLINE
   nomfic = input("entrez un nom de fichier : ")
   os.write(writefd, nomfic)   # envoyer nom fichier sur tube vers serveur
   # lire contenu du fichier sur tube depuis serveur
   buff = os.read(readfd, MAXLINE)
   while (len(buff) > 0):
       os.write(1, buff)  # ecrire contenu du fichier sur sortie standard
       buff = os.read(readfd, MAXLINE)
```

```
#! /usr/bin/env python
# file: server.py (module server)

import sys,os

def server(readfd, writefd):
    global MAXLINE
    buff = os.read(readfd, MAXLINE)
    try:
        fd = os.open(buff, os.O_RDONLY)
    except:
        os.write(writefd,buff)
        sys.exit(1)

    # ouverture reussie
    buff = os.read(fd, MAXLINE)
    while (buff.len > 0):
        os.write(writefd,buff)
        buff = os.read(fd, MAXLINE)
```

```
os.close(fd)
```

2.2 Tubes nommés, ou FIFOs

Un tube nommé, ou FIFO, est utilisé comme un tube ordinaire, mais il a un nom dans le catalogue, comme un fichier ordinaire. On peut par exemple créer les FIFOs, par convention, dans le catalogue `/tmp`.

La commande

```
import os
os.mkfifo(nomfic, mode)
```

crée un FIFO avec le nom symbolique `nom` et le mode d'accès `mode`. On peut ainsi faire fonctionner un FIFO entre deux processus indépendants (par exemple le serveur étant lancé en tâche d'arrière-plan).

On peut dès lors lancer indépendamment un serveur et un client, et on n'a plus besoin de programme principal. Il suffit que le client et le serveur s'accordent sur le nom des FIFOs qu'ils utilisent.

2.3 Question 1

Dans un premier temps, modifier le programme `mainpipe.py` pour faire communiquer un client et un serveur par FIFOs, le serveur étant toujours fils du client. A-t-on besoin de modifier aussi les programmes du client et du serveur ?

2.4 Question 2

On demande de modifier les programmes de la section 2 pour construire un client et un serveur indépendants communiquant par FIFO, en supprimant le programme principal.