Erick Gallesio, Stéphane Lavirotte, Jean-Yves Tigli, Hélène Collavizza
Polytech'Nice – Sophia
SI3
2018-2019

TD n°6
Redirections et Tubes

Ces exercices mettent en œuvre les mécanismes de redirection de Posix (primitives `fcntl()`, `dup()`, `dup2()`), ainsi que la communication de données entre processus par tube (primitive `pipe()`) et tube nommé (lecture/écriture dans un fichier spécial). Nous aborderons aussi sur la fin du TD les similarités et différences entre Unix et Windows pour les tubes.

Pour réaliser ce TD, nous vous fournissons une partie du code et un `Makefile` pour vous permettre de vous concentrer sur les parties importantes. Vous pouvez récupérer l'archive à l'adresse suivante :

`http://trolen.polytech.unice.fr/cours/progsys/td06/td06_distrib.zip`

1 Redirections

Les redirections sont un moyen d'envoyer des données produites par un processus dans un fichier ou de récupérer de données stockées dans un fichier à un processus. L'entrée standard, la sortie standard et la sortie standard d'erreur ainsi que les fichiers à partir desquels on accède aux données ou dans lesquels on envoie les données sont tous gérés par des descripteurs de fichier. Le mécanisme de duplication des descripteurs sera donc le mécanisme utilisé pour mettre en œuvre les redirections sous Unix.

Exercice n°1:

Ecrire un programme `redir_simple` permettant de faire la redirection de la commande `ls -l` vers le fichier `foo`.

Exercice n°2:

Ecrire le programme `redir` permettant de rediriger les résultats d'une commande Unix de (ou vers) un fichier. Notre programme aura comme premier paramètre soit le caractère 'R' pour faire une redirection en lecture, soit le caractère 'W' pour une redirection en écriture. Le second paramètre contient toujours le nom du fichier qui est la source ou la destination de la redirection. L'ensemble des paramètres suivants correspond à la commande (commande, options et arguments) qui doit être lancée par le programme `redir`.

Des exemples d'utilisation sont donnés ci-dessous :

- `redir R /etc/passwd cat -n` permet d'afficher le contenu du fichier `/etc/passwd` (l'option `-n` de la commande `cat` permettant de numéroter les lignes).
- `redir W date.txt date` écrit la date dans le fichier `date.txt`
- `redir W out.txt cat -n /etc/passwd` copie le contenu du fichier `/etc/passwd` dans le fichier `out`.

Le fichier `tst_redir.sh` vous permettra d'évaluer rapidement si votre programme fonctionne avec ces exemples.

> **Suggestion** : Pour simplifier l'exécution de la commande passée en paramètre avec toutes les options et arguments, vous utiliserez la primitive `execvp(const char* cmd, char *const argv[])`.

2 Tubes

2.1 Introduction

Un tube est un moyen de transmission de données d'un processus à un autre. C'est une des méthodes de communication interprocessus (« *Inter Process Communication* » : IPC).

Un tube a les particularités suivantes :

- La communication est unidirectionnelle: on écrit à un bout et on lit à l'autre (d'où le nom de tube). Cela implique qu'il faut au moins deux descripteurs pour manipuler un tube.
- La communication est faite en mode FIFO (First In First Out), premier écrit, premier lu. Des primitives comme `lseek()`, ou tout autre accès directe à un donnée, n'ont pas de sens pour un tube.

- Ce qui est lu quitte définitivement le tube et ne peut être relu. De même, ce qui est écrit est définitivement écrit et ne peut être enlevé.
- La transmission est faite en mode flot continu d'octets. L'envoi consécutif des deux séquences « `abcd` » et « `efg` » est semblable à « `abcdefg` » et peut être lu en totalité ou en morceaux comme « `ab` », « `cde` » et « `fg` » par exemple.
- Pour fonctionner, un tube doit avoir au moins un lecteur et un écrivain. Il peut y en avoir plusieurs.
- Un tube a une capacité finie, et il y a une synchronisation type producteur / consommateur entre lecteurs et écrivains: un lecteur peut parfois attendre qu'il y est quelque chose d'écrite avant de lire, et un écrivain peut attendre qu'il y ait de la place dans le tube avant de pouvoir y écrire.

2.2 Tube Anonyme (*pipe*)

2.2.1 Création d'un tube anonyme

La création d'un tube anonyme est réalisée grâce à la primitive `pipe()` qui utilise deux descripteurs de fichier (un tableau de deux entiers).

```
#include <unistd.h>
int pipe(int p[2]);
```

Un tube correspond donc à la description de `2` descripteurs de fichiers, l'un permettant d'écrire dans le tube (`p[1]`), l'autre permettant d'y lire (`p[0]`).

2.2.2 Utilisation d'un tube anonyme

On utilise alors les opérations classiques de lecture et d'écriture (primitives `read()` et `write()`).

Si un processus père crée un tube et qu'il fait ensuite un appel à `fork()` pour créer un processus fils, comme le fils est l'exacte copie du père, il héritera lui aussi du tube. Il disposera donc des mêmes descripteurs en lecture/écriture aux extrémités du tube. Pour que deux processus puissent communiquer des données par un tube, il faut être le descendant d'un même père (ou ancêtre commun) qui crée le tube. Ce dernier peut lui-même être l'un des processus communiquant.

Remarque :
- La seule façon de placer une fin de fichier dans un tube est de fermer celui-ci en écriture dans tous les processus qui l'utilisent (`close(p[1])`).
- Si un tube est fermé en lecture par tous les processus qui l'utilisent et qu'un processus tente d'écrire dedans alors ce dernier recevra le signal `SIGPIPE` qui, s'il n'est pas capturé, interrompt le processus.

La primitive `read` renvoie `0` lorsque la fin de fichier est atteinte, c'est-à-dire si le tube est vide et si tous les processus qui utilisent ce tube l'ont fermé en écriture.

Exercice n°3:

Ecrire un programme `tube_anonyme1.exe` qui crée un tube et y écrit `10` caractères ("`0123456789`" en une seule écriture). Le programme appelle ensuite une fonction `void lecture(int fd)` qui lit le contenu du tube caractère par caractère jusqu'à la fin de fichier. Pour faire l'affichage des caractères lus dans `lecture`, vous utiliserez aussi la fonction `write` sur le descripteur de fichier de la sortie standard.

Pour cette première version, un seul processus sera mis en jeu et sera à la fois écrivain et lecteur du tube. Si cette version a peu d'intérêt sur le plan pratique, elle aura pour mérite de mettre en œuvre la structure de tube au sein de votre programme et de vous rendre compte de la nécessité de fermer correctement et au bon moment les descripteurs de fichier.

TD n°6
Redirections et Tubes

Exercice n°4:

Ecrire un programme `tube_anonyme2.exe`. Pour cette seconde version, le processus père sera l'écrivain dans le tube et le processus fils fera appel à la fonction `lecture` mise en place précédemment. Vous veillerez dans le père à faire une pause de `2` secondes pour constater que, tant que le tube n'est pas fermé, le fils est en attente car il ne reçoit pas `EOF` (End Of File).

Exercice n°5:

Quand on utilise un tube, on n'est pas limité à un seul écrivain ou un seul lecteur. Ecrire un programme `tube_anonyme_multi.exe` qui aura deux fils qui écriront dans le tube et le père qui lira les données. Les deux fils écrivent respectivement l'alphabet l'un en majuscule (« ABCD…YZ ») et l'autre en minuscule (« abcd…yz »). Les écritures se feront `2` caractères par `2` caractères et une pause d'une seconde sera faite entre deux écritures par chacun des fils. Le père tentera de lire `3` caractères au maximum dès que possible.

Modifiez le temps pour que le premier fils écrive toutes les `1` secondes alors que le second écrira toutes les `2` secondes.

> **Remarque** : Vous pourrez noter que :
> - L'écriture est atomique ; chaque séquence majuscule est écrite en bloc.
> - Les blocs écrits s'enchaînent entre eux « de façon quelconque », les écrivains agissant indépendamment l'un de l'autre (EFGH montre que le premier fils a écrit deux fois de suite avant l'autre). Ceci est bien évidemment dû à la pause entre deux écritures simulant un traitement d'instructions dans ce laps de temps.
> - Le lecteur ne peut pas distinguer entre les deux écrivains sans un « protocole » établi avec eux. Un tel protocole pourra être une taille fixe des blocs messages pour tous les processus, et chaque message est précédé de leur identité pour les processus écrivains (e.g. leur *pid*).

2.3 Tube Nommé (*named pipe* ou *FIFO*)

Une des limitations fortes des tubes anonymes est que la communication ne peut s'établir qu'entre des processus qui sont apparentés (un fils et son père ou son ancêtre qui l'a créé). Si l'on souhaite communiquer entre n'importe quel processus sur une même machine, il faut alors utiliser les tubes nommés.

Les tubes nommés allient les propriétés des tubes standards et celles des fichiers. Comme les tubes ordinaires, les tubes nommés ont les mêmes propriétés que celle énoncées dans l'introduction à savoir : taille limitée, lectures et écritures en mode `FIFO`, informations lues obligatoirement extraites du tube et pas d'opérations de type `seek`.

2.3.1 Création d'un tube nommé

Un tube nommé peut être créé par n'importe quel processus par `mkfifo` ou la fonction plus générique `mknod`, (dont la fonction est de créer des dossiers, des fichiers spéciaux ou ordinaires).

```
#include <sys/stat.h>
int mkfifo(char *nom, mode_t mode);
int mknod(const char *ref, mode_t mode, dev_t droits)
```

Il est bien sûr possible de créer un fichier spécial de tube nommé à partir de commande Shell : `mkfifo` ou `mknod`.

2.3.2 Utilisation d'un tube nommé

Tout processus qui veut communiquer par un tube nommé doit connaître son nom (donc le nom du fichier spécial). Un processus doit alors ouvrir par `open` un de ses descripteurs selon le type d'opération qu'il doit faire (lecture ou écriture). Les entrées/sorties sur un tube nommé, se passent comme pour les tubes anonymes. `read()` et `write()` sont bloquants ou non selon le mode d'ouverture effectuée. Ce statut peut changer par appel à `fcntl()`. En général, un processus serveur choisira une ouverture en écriture non bloquante pour pouvoir écrire des messages sans attendre. Un processus client, ouvrira un tube nommé en lecture bloquante pour attendre de lire un message. Si un

processus essaie d'ouvrir un tube nommé en lecture il sera suspendu jusqu'à ce qu'il existe un processus qui ouvre ce tube en écriture (et vice versa).

Exercice n°6:

A l'aide de la commande Shell `mkfifo`, créez un tube nommé `my_named_pipe` dans `/tmp`. Un premier programme `ecrivain.exe` écrira dans un tube nommé et un programme `lecteur.exe` consommera les données injectées par `ecrivain.exe`. Le nom du tube nommé utilisé par l'écrivain et le lecteur sera passé en paramètre au lancement des programmes.

Vous venez de mettre en œuvre un système de chat unidirectionnel (de l'écrivain vers le lecteur).

Question 1: Que faudrait-il faire pour avoir un système de chat symétrique donc bidirectionnel, c'est-à-dire que chaque processus soit à la fois lecteur et écrivain ?

Un système de chat sur une même machine a peu d'intérêt (sauf pour deux utilisateurs connectés à distance à la même machine). Nous aimerions bien sûr pouvoir communiquer entre processus sur des machines distantes. Pour tout savoir sur ces possibilités, passez à la section suivante !

Question 2: Que se passe-t-il s'il y a plusieurs lecteurs sur un tube nommé (vous constateriez le même comportement avec des tubes anonymes) ? Expliquez le phénomène observé à partir de vos connaissances.

3 Et si on veut communiquer entre processus « distants »

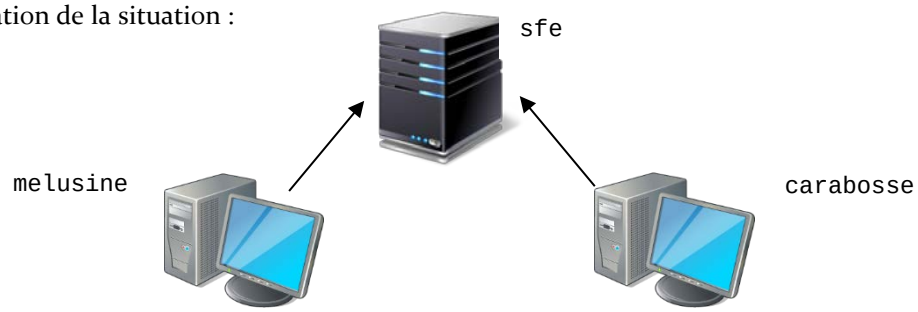
Le but de cette section est de prendre un peu de recul et de se demander si on pourrait faire une communication entre processus sur des machines distantes reliées en réseau.

3.1 Communication par tube nommé entre machines sous Linux

Dans le cas de l'utilisation des tubes nommés, il faut que les processus utilisent un fichier spécial qui leur soit accessible à tous deux. Dans l'exercice précédent, c'était le cas car nous étions sur la même machine. Mais sur deux machines différentes en réseau, comment faire ?

En théorie, il faudrait simplement que les deux machines aient accès à un même système de fichiers partagé et le tour devrait être joué ! Nous disposons à Polytech de ce type d'infrastructure pour tester cela rapidement. En effet, vous pouvez avoir accès à votre compte étudiant qui est stocké sur la machine `sfe` via la machine `melusine` ou la machine `carabosse`. En fait la machine `sfe` peut être considérée comme un NAS et les deux machines `melusine` et `carabosse` montent le même système de fichiers à l'aide du protocole NFS (Network File System).

Voici une illustration de la situation :



Exercice n°7:

Copiez les fichiers `ecrivain.c` et le `Makefile` sur votre compte sur la machine `melusine.polytech.unice.fr`.

```
scp ecrivain.c Makefile user@melusine.polytech.unice.fr:
```

Faites la même chose avec les fichiers `lecteur.c` et le `Makefile` sur vote compte, mais sur la machine `carabosse.polytech.unice.fr`.

```
scp lecteur.c Makefile user@carabosse.polytech.unice.fr:
```

A l'aide de deux terminaux, connectés vous sur chacune des machines :

- Sur la machine `melusine`, lancez la commande : `make ecrit`. Cet appel compilera votre programme, créera le tube nommé et lancera le programme `ecrivain`.
- Sur la machine `carabosse`, lancez la commande : `make lit`. Cet appel compilera votre programme, vérifiera que le tube existe bien et lancera le programme `lecteur`.

Testez la communication. Vous avez un problème Houston ? Eh bien oui, cela ne doit pas marcher… Mais pourquoi ? En fait, la communication entre processus par tube nommé sous Linux est implémentée à l'aide de mémoire partagée dans le noyau. Or les processus ne tournent pas sur la même machine et donc ne peuvent pas partager de mémoire. Donc chaque machine voit seulement l'écrivain ou le lecteur, mais pas les deux, même en accédant au même fichier.

Donc c'est fichu, on ne pourra pas faire de communication par tube nommé entre des machines… Et bien si, mais si on change de système (et oui c'est dépendant de l'implémentation du système d'exploitation).

3.2 Communication par tube nommé entre machine sous Windows

Le noyau Windows NT avec son API Win32 propose des similarités avec les tubes Unix/Posix. Mais il propose aussi des mécanismes beaucoup plus complets autour de la notion de tubes. Si l'on retrouve les tubes anonymes et les tubes nommés, des différences notables sont à souligner.

Par exemple, les tubes nommés fournissent des communications de données unidirectionnelles et bidirectionnelles entre des processus sur le même ordinateur ou entre les processus sur différents ordinateurs sur un réseau. Le développement d'applications utilisant des tubes nommés est en fait assez simple et ne nécessite aucune connaissance formelle des protocoles de transport réseau sous-jacents (tels que TCP/IP). Cela est dû au fait que les tubes nommés utilisent le redirecteur Microsoft Network Provider (MSNP) pour établir la communication entre les processus sur un réseau, cachant ainsi les détails du protocole réseau de l'application.

Voici un tableau de synthèse des avantages et inconvénients de méthodes pour réaliser des IPC sous Windows.

Technology	**What is it**	**Advantages**	**Disadvantages**
Anonymous Pipes	Uses shared memory for communication. Lets two processes transfer information in one-way synchronously	• Less overhead that named pipes	• Both process must be on the same machine • On-way communication only • Synchronous communication only
Names Pipes	Uses shared memory for communication. Lets two processes transfer information one-way or both directions, synchronously or asynchronously	• Both processes can be on the same machine or networked machines • One-way and bidirectional communication • Synchronous and asynchronous communication	

TD n°6
Redirections et Tubes

Mailslots	A process can broadcast datagram messages to many other processes	• Works across a network • Datagrams can be broadcasted over network	• Messages can get lost between sender and receiver • No way for sender to know messages was received

Table 1: Technologies "Inter Process Communication" sous Windows[1]

Exercice n°8:

Pour tester la mise en œuvre des tubes nommés sous Windows entre machines en réseau, nous vous fournissons le code dans la solution `Win32NamedPipe`. Cette solution contient 3 projets : une bibliothèque contenant du code commun pour la gestion des opérations de base sur les tubes nommés, et deux projets pour un serveur et un client communicant avec les tubes nommés. Commencez par tester ces programmes sur votre machine. Puis, « modifiez le code » pour tester la configuration suivante : le serveur sur votre machine et le client est lancé depuis la machine d'un binôme (en spécifiant le nom de la machine dans le chemin d'accès au tube nommé par le client). Dans le cas où cela fonctionne bien, modifiez le programme pour faire en sorte que le nom du tube nommé soit passé en paramètre au lancement du client.

3.3 Communication entre processus à tous les étages

Même si cette dernière solution fonctionne, il serait illusoire de penser utiliser une telle solution en dehors d'un Intranet (pour des raisons de sécurité entre autre). Mais comment communiquent des processus entre des machines à plus large échelle ? Et bien avec des `socket` bien sûr ! Vous avez étudié cela dans le cours « Internet et Réseaux ». Nous reviendrons sur cela un peu plus tard pour mettre en relation ces deux cours.

Vous serez aussi amenés à découvrir d'autres approches ou technologies sont utilisées pour réaliser des communications interprocessus, comme les « *Remote Procedure Call* » (RPC) et les Web Services par exemple. Mais c'est pour plus tard, en SI4 entre autre.

[1] http://www.drdobbs.com/windows/using-named-pipes-to-connect-a-gui-to-a/231903148

TD n°6
Redirections et Tubes

Pour aller plus loin

Voici des exercices qui mélangent l'utilisation des redirections, des tubes et la création de processus.

Exercice n°9:

Le but de cet exercice est d'implémenter un équivalent de la fonction `popen` en lecture seule. D'après la page de manuel (`man 3 popen`), cette fonction permet de récupérer les données produites par un processus dans le processus appelant. Vous réaliserez une fonction `mypopen` simplifiée qui créera un tube, invoquera un Shell (`execXX`) pour exécuter une commande passée en paramètre et retournera le descripteur de fichier permettant d'accéder aux données produites par la commande exécutée. Votre programme principal utilisera cette fonction pour afficher les données produites par la commande et se terminera en affichant le nombre de caractères total lus dans le tube.

Exercice n°10:

Écrire un programme, `tst_pipes.exe`, qui simule la commande au Shell suivante :

```
ps ax | grep bash | wc –l
```

On lancera ici 3 processus et donc 2 tubes. En écrivant ce programme, prenez grand soin de fermer, dans tous les processus, les descripteurs de fichiers inutilisés, en particulier ceux des tubes. Mais il est aussi intéressant de faire l'expérience de ne pas les fermer. Que se passe-t-il alors ? Pourquoi ?

Exercice n°11: Crible d'Eratosthène

Utiliser un système de tubes pour implémenter l'algorithme du crible d'Eratosthène. Votre programme prendra en paramètre le nombre d'entiers à énumérer. Pour chaque nouveau nombre premier trouvé, le programme affichera ce nombre et créera un nouveau processus chargé d'éliminer ses multiples. La communication entre les différents filtres sera réalisée au moyen de tubes. Là encore, on vérifiera que le programme ne laisse pas « traîner » des processus derrière lui (en fermant bien les tubes correctement).