Programmation Système

IPC &Tubes

Abdelkader Gouaïch

2022

PARTIE I: INTRODUCTION AUX IPC

Objectif: coordination des activités entre des processus indépendants.

Coordination = Communication

Communication = synchronisation + partager des données

L'ensemble de ces techniques est: *Inter Process Communication (IPC)*

IPC vont nous permettre:

- envoyer des signaux
- synchroniser les activités entre processus
- transférer des données entre processus

IPC: LES SIGNAUX

Déjà vu !😎

Les signaux:

- connaître un changement d’état de l’environnement
- échanger des informations élémentaires (entiers)

IPC: LA SYNCHRONISATION

=> Coordination des activités

Objets pour le faire:

- les sémaphores entre processus
- les mutex entre les threads
- les primitives d'attente comme `wait` et `join`.

IPC: LA COMMUNICATION

Cette catégorie regroupe les mécanismes pour échanger des structures de données.

Les modalités d'échange sont:

- le transfert direct d'octets en utilisant soit un flux ou des messages;
- le partage d'un segment mémoire entre les mémoires virtuelles des processus.

Transfert de données: flux ou message

flux:

- les processus envoient et reçoivent les octets sur un canal de communication.
- Ce canal est semblable à un fichier: nous pouvons écrire et lire des blocs d'octets avec un ordre de lecture qui respecte celui de l'écriture (First In First Out)

Exemples:

- les tubes, les tubes nommés
- les sockets TCP.

Transfert de données: flux ou message

Messages:

- une unité d'échange, appelée *le message*.
- Le message sera transféré entre les différents processus en utilisant un protocole de communication.

Les messages sont échangés et rangés des boîtes à lettres à la réception.

L'ordre d'arrivée des messages dans ce modèle n'est pas nécessairement celui des envois.

Exemples:

- les files de messages et les sockets UDP.

Communication par mémoire partagée

Une autre façon de communiquer consiste à partager tout simplement un segment mémoire.

=> partager des variables !

Exemples:

- shared memory
- memory mapping

PARTIE II: INTRODUCTION AUX TUBES UNIX

```
ls | grep '.txt'
```

Cette commande :

- va créer un tube
- le flux sortant de `ls` => le flux entrant de `grep`.
- lister uniquement les fichiers avec le suffixe ‘.txt’

LES TUBES SONT ORIENTÉS FLUX

Un flux de données (octets):

- La communication se fera par des échanges d'octets.
- Le nombre d'octets échangés lors d'une session n'est pas limité et pas forcément connu par le destinataire.
- L'ordre des octets écrits est préservé lors de la lecture.

Un flux ~ un fichier standard

- read, write, close
- pas de `lseek()`.

LECTURE DANS UN TUBE ET LA FIN DE COMMUNICATION

- Une demande de lecture dans un tube vide va bloquer le lecteur.
- Si l'extrémité d'écriture du tube est fermée alors le lecteur va lire la constante `EOF` (End Of File)
- Aucun écrivain n'est présent => communication terminée.

LE TUBE EST UNIDIRECTIONNEL

Les tubes sont unidirectionnels:

- Il existe un sens de transfert des données qui se fera toujours de l'extrémité d'écriture vers l'extrémité de lecture.
- Les processus se trouvant aux extrémités doivent alors prendre des rôles d'écrivain (émetteur) et de lecteur (destinataire)

L’ÉCRITURE CONCURRENTE SUR UN TUBE

Si plusieurs processus partagent le même tube alors il est possible d'écrire jusqu'à n < PIPE_BUF octets atomiquement

Pas de problème de synchronisation !

LA TAILLE DE MÉMOIRE TAMPON D’UN BUFFER

Un tube possède une mémoire tampon de taille limitée.

Si la mémoire tampon est atteinte alors toutes les nouvelles tentatives d’écriture seront bloquées en attendant que les lecteurs consomment les octets déjà présents.

LES TUBES ANONYMES

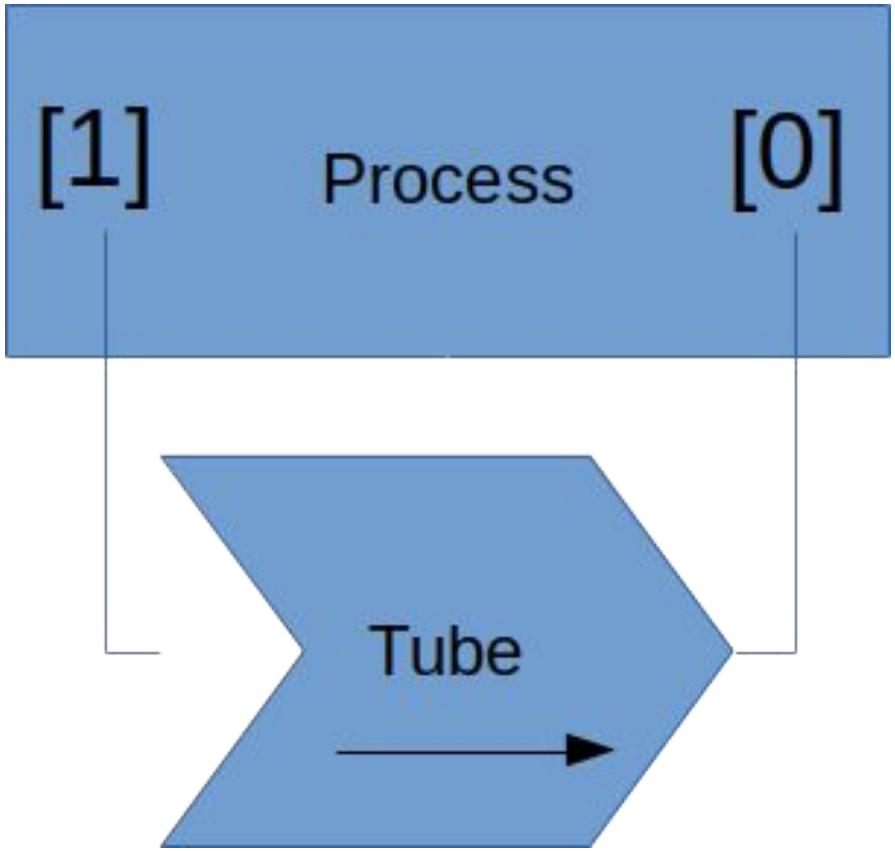
Création et utilisation d'un tube anonyme

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

La fonction `pipe()` prend en paramètre un tableau d'au moins deux entiers.

- la case d'index 0 descripteur *l'extrémité lecture* du tube.
- la case d'index 1 descripteur *l'extrémité d'écriture* du tube.

Modèle de flux de données



Modèle de fonctionnement d'un tube

COMMENT UTILISER LES TUBES ?

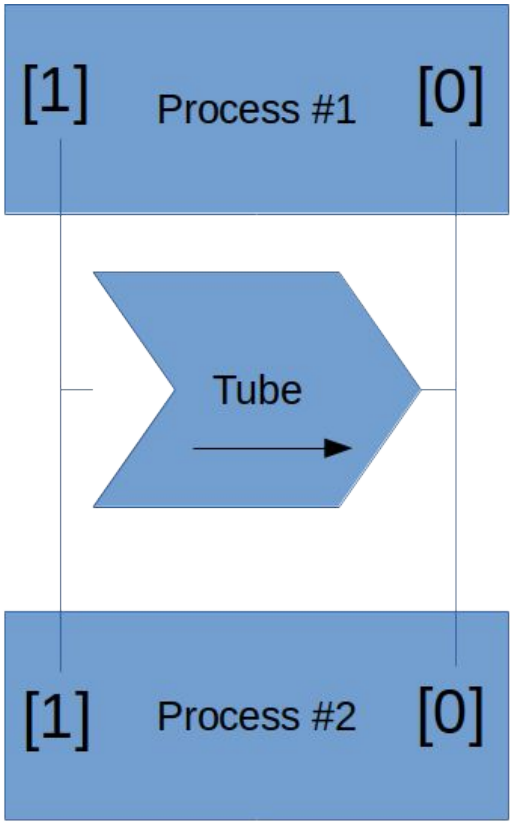
Un tube créer un flux directionnel d’octets entre ses deux extrémités.

Si un seul processus a connaissance du tube alors le bénéfice est limité: le processus, va écrire et lire ses propres données. L’utilisation du tube sera dans ce cas similaire à l’utilisation d’une mémoire tampon annexe.

L’intérêt des tubes se révélera dans le cas où au moins deux processus partagent le *même* objet tube

- un processus va écrire dans l’extrémité write
- un processus va lire depuis l’extrémité read

Ainsi, nous allons établir un canal directionnel de communication entre des processus différents pour échanger des octets.

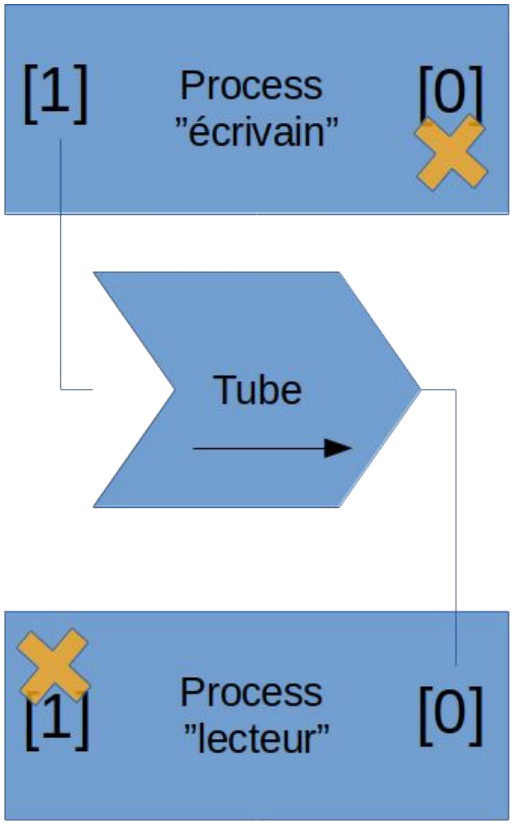


Modèle de fonctionnement d’un tube avec deux processus

Nous savons que le tube offre un flux directionnel.

Chaque processus va explicitement prendre son rôle:

- Fermer une des extrémités qui ne sera pas utilisée.
- Le processus écrivain va fermer l'extrémité de lecture e
- Le processus lecteur va fermer l'extrémité d' écriture



Modèle de fonctionnement d’un tube avec deux processus qui prennent des rôles écrivain et lecteur

Cette configuration offre aussi l’avantage de gérer correctement la fin de la communication entre les processus.

COMMENT RÉALISER UNE COMMUNICATION BI-DIRECTIONNELLE ?

Si nous souhaitons établir une communication bi-directionnelle entre deux processus: créer deux tubes avec des sens opposés.

Un protocole applicatif devra spécifier comment l'échange des données se fera entre ces deux processus.

TECHNIQUE DE DUPLICATION DES DESCRIPTEUR

`dup2(des,l_des)` qui va prendre deux paramètres et réaliser les opérations suivantes:

- l'ancien descripteur `l_des` sera fermé
- la valeur de `l_des` sera récupérée et liée maintenant au fichier identifié par `des`

Dans le cas des tubes, `dup2` sera utilisée de la façon suivante:

```
dup2(tube[1], 1);
```

Ce code ferme le fichier de sortie standard (lié au terminal par défaut) et associe le descripteur de valeur 1 à l'extrémité d'écriture de notre tube.

Exemple:

```
dup2(tube[1], 1);
close(tube[1]);
printf("Bonjour à vous !");
```

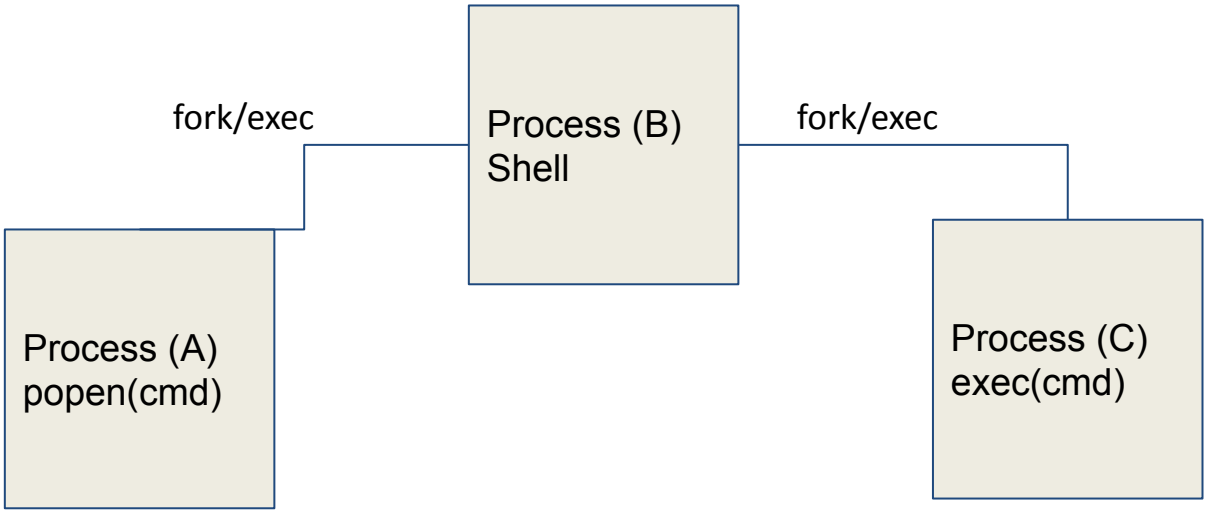
`printf` va écrire le message dans le tube car nous avons lié la sortie standard avec l'extrémité d'écriture du tube.

LES TUBES PROCESSUS

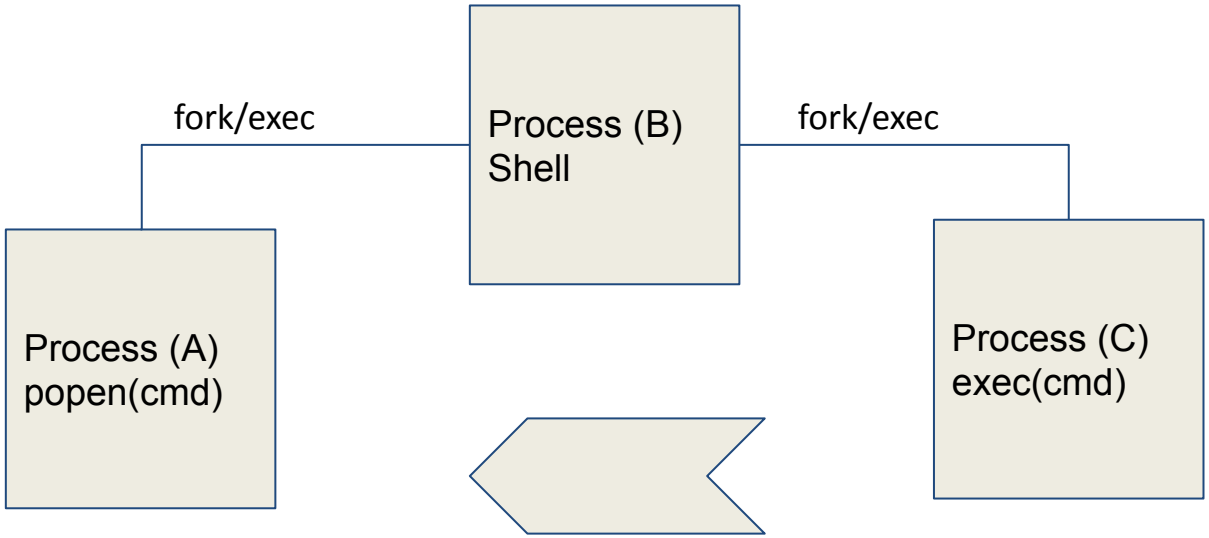
```
#include <stdio.h>
FILE *popen(const char * command ,
const char * mode );
int pclose(FILE * stream );
```

popen:

- Le processus appelant (A), va lancer un processus shell (B), avec fork/exec
- Le processus shell (B) va réaliser un fork/exec dans un nouveau processus noté (C)
- processus (C) va exécuter la commande

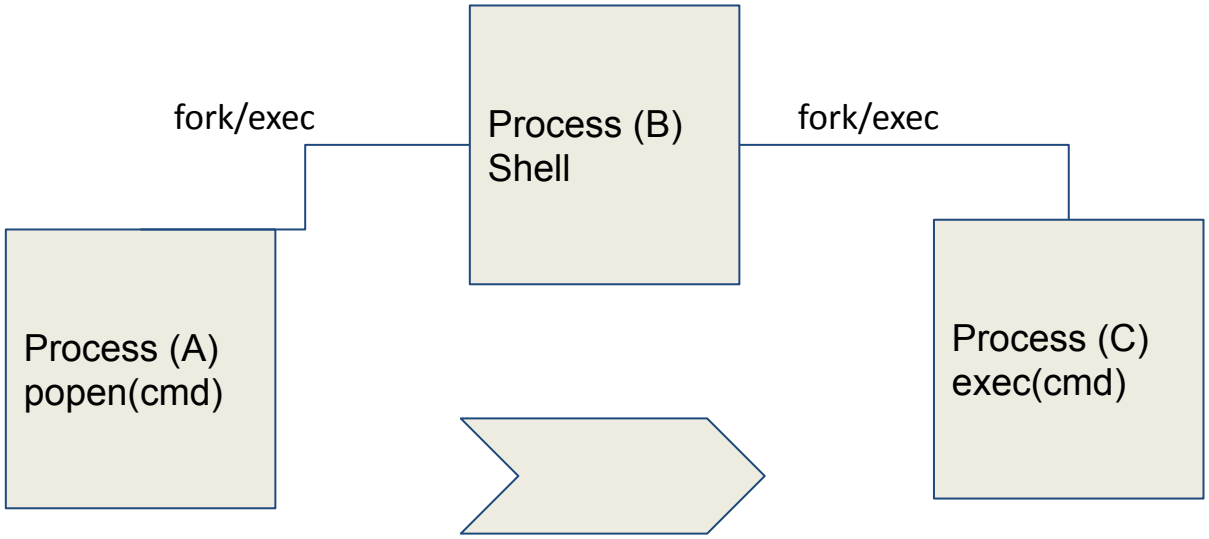


MODE R(EAD)



Le premier cas est le mode lecture avec le paramètre `mode` qui prendra la valeur `R`. Dans ce cas, un tube directionnel fera le lien entre le processus (C) et le processus (A).

MODE W(RITE)



Le processus (A) sera capable de transmettre des informations au processus (C) qui seront lues sur son entrée standard.

LES TUPES NOMMÉS

INTRODUCTION

Un tube nommé, ou FIFO:

- un tube identifié
- des processus qui partagent le nom de la FIFO peuvent l'ouvrir et communiquer.

Utilisation des FIFO:

- comme un simple fichier avec les opérations de lecture, écriture et fermeture.

FIFO:

- mode de transmission des octets en flux
- respecte l'ordre d'écriture

CRÉATION DE TUBES NOMMÉS

Méthode par le shell

La commande shell `mkfifo`

```
mkfifo /tmp/montube
```

`ls` va lister les tubes avec un flag '`p`':

```
prw-rw-r-- 1 u u /tmp/montube
```

Méthode par API C

```
#include <sys/stat.h>
int mkfifo(const char *
pathname , mode_t mode );
```

`mkfifo()`:

- créer une fifo à partir du code C.
- Le mode va spécifier les droits associés à la fifo comme pour un fichier standard.

UTILISATION D’UN TUBE NOMMÉ

Ouverture bloquante

Nous devons spécifier notre rôle par rapport à la FIFO à l'ouverture.

Cela est possible en spécifiant le mode d'ouverture soit en lecture ou écriture avec les flags `O_RDONLY` et `O_WRONLY`.

Ouverture bloquante

L'ouverture d'une FIFO en mode lecture avec `O_RDONLY`

- va donc bloquer le processus qui en fait la demande jusqu'à la présentation d'un processus qui va ouvrir la FIFO en mode écriture avec le flag `O_WRONLY`.

De façon similaire, l'ouverture d'une FIFO en mode écriture avec le flag `O_WRONLY` sera bloquante jusqu'à la présentation d'un processus qui ouvrir la FIFO en lecture avec le flag `O_RDONLY`.

Ouverture bloquante

L'ouverture d'une FIFO en mode écriture avec le flag `O_WRONLY` sera bloquante jusqu'à la présentation d'un processus qui va ouvrir la FIFO en lecture avec le flag `O_RDONLY`.

EXEMPLE D’UTILISATION DE FIFO

La commande `wc` avec l'argument `-l` permet de comptabiliser le nombre de lignes dans un flux texte.

```
$wc -l
>premiere
>deuxieme
>troisieme
^D
$3
```

Nous allons créer une FIFO nommée `testwc` sous le répertoire /tmp:

```
$mkfifo /tmp/testwc`
```

Nous lançons ensuite le processus `wc` en background avec une redirection de son entrée standard vers la FIFO:

```
$wc -l < /tmp/testwc &
```

Nous notons que ce processus est en cours d'exécution dans le groupe background : il est bloqué en attente d'un écrivain.

```
$ls -al > /tmp/testwc
```

Le processus `ls` va écrire son résultat sur la FIFO.

Le texte sera récupéré par le processus `wc` qui attend en background et qui affichera le nombre de lignes.