Communications inter-processus sous Unix

Hafid Bourzoufi

Didier Donsez

Université de Valenciennes

Institut des Sciences et Techniques de Valenciennes

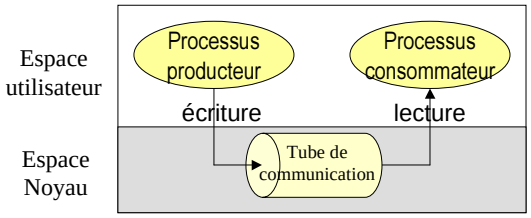
`donsez@univ-valenciennes.fr`

Les différents moyens de communication sous Unix

- Communication entre processus
  - d'un même noyau (i.e. même machine)
    - Signaux
    - Tubes de communication
    - Tubes de communication nommés
    - IPCs System V
      - les files de message
      - les mémoires partagées
      - les sémaphores
  - sur des noyaux séparés (voir même non-Unix)
    - Sockets BSD (TCP/IP)
    - TLI

Tubes de communications

- Un tube a une structure FIFO
  - Les communications sont unidirectionnelles
  - utilise les primitives des fichiers
    - read, write, close, ...



Création des tubes de communications

- Primitive de création d'un tube

```
int pipe (int p[2])
```

  crée 2 descripteurs de fichier
  - p[0] : permet de lire dans le tube
  - p[1] : permet d'écrire dans le tube
- Retour
  - 0 si la création du tube a été correctement effectuée
  - -1 en cas d'erreur.
    - trop de fichiers ouverts par le processus appelant la table des fichiers du système est pleine
    - erreur dans le paramètre

Généralités sur les tubes de communication

- Un tube de communication a une taille limitée
  - L'écriture dans un tube plein est bloquante
- Une écriture dans un tube sans lecteur potentiel provoque une erreur
  - le processus reçoit le signal SIGPIPE
- La fin de fichier dans un tube est atteinte si celui-ci est vide et s'il n'existe pas de processus susceptibles d'y écrire
  - tous les processus ont fermé le tube en écriture
- Limitation
  - Un tube ouvert par un processus ne peut être utilisé que par les processus descendants du processus qui a créé le tube
    - car héritage des descripteurs ouverts

Utilisations des tubes de communication

- Pour faire communiquer deux processus à l'aide d'un tube de communication, les actions suivantes doivent être respectées :
  - Un processus (père) ouvre un tube de communication par la primitive (pipe)
  - Il crée ensuite l'autre processus par la primitive fork()
    - les deux processus partageront les deux descripteurs de fichier
  - Pour envoyer une message, le processus écrit dans le tube à l'aide de la primitive write()
  - Pour recevoir un message, le processus lit dans le tube à l'aide de la primitive read()
  - Les processus ferment le tube à l'aide de la primitive close() : close(p[0]), close (p[1])

Remarques générales sur les tubes de communication

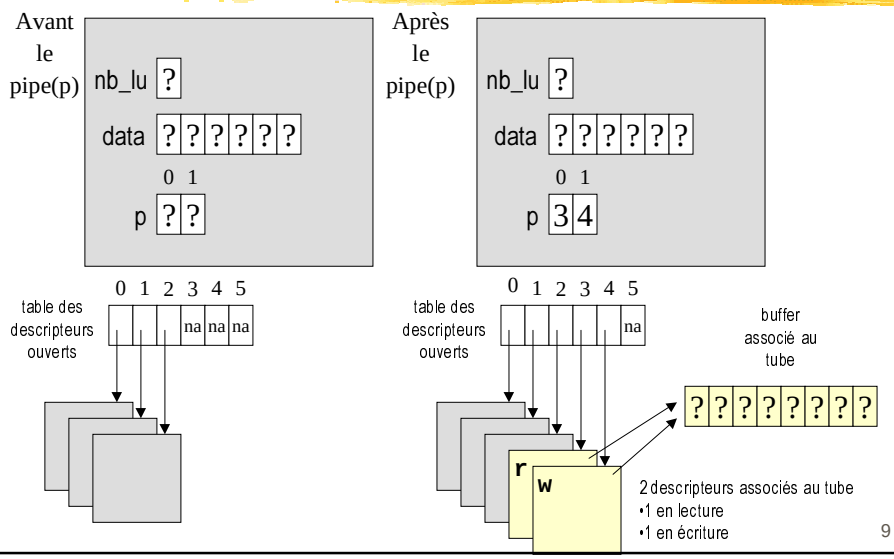
- Modèle Producteur-Consommateur
  - en général, les applications obéissent au modèle producteur/consommateur
  - le père consomme
  - le fils produit

Exemple d'utilisation des tubes de communications avec le fork()

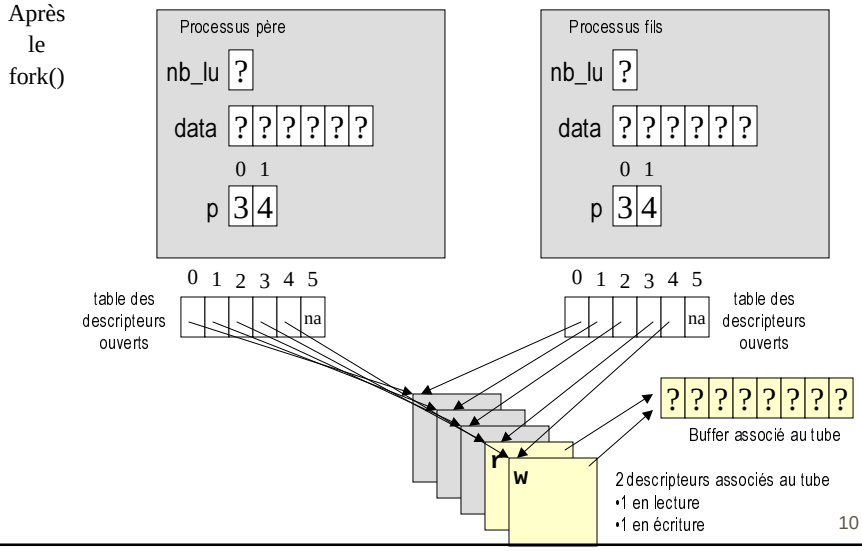
```
#include<stdio.h>
#define MAX 6
main()
{ int p[2],nb_lu;
  char data[MAX];
if (pipe(p)==-1){
        perror("pipe");exit()};
```

```
if (fork()==0){/* le fils produit*/
   close(p[0]);
   nb_lu=read(0,data, MAX );
   write(p[1],data,nb_lu);
   close(p[1]);
   wait();
} else { /* le père consomme */
   close(p[1]);
   nb_lu=read(p[0],data, MAX/2 );
   write(1,data,nb_lu);
   close(p[0]);
   exit();
}
}
```

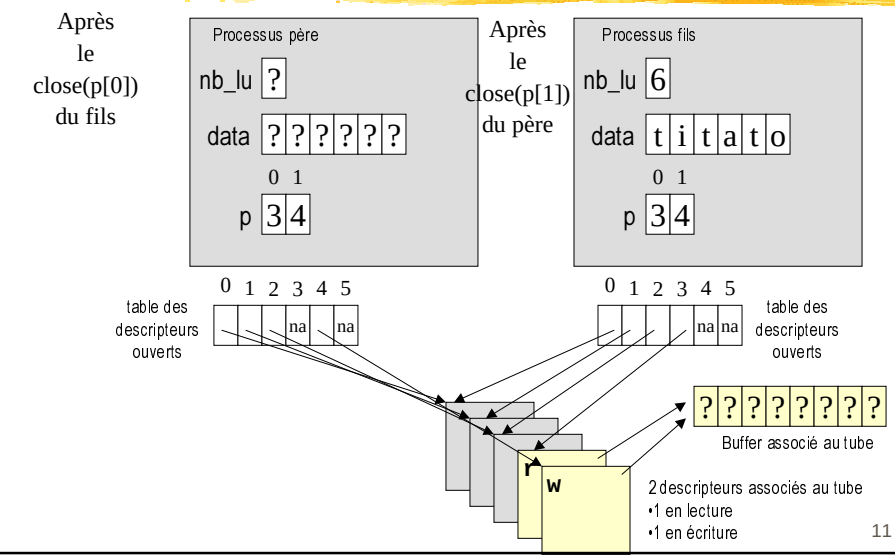
Exemple d'utilisation des tubes de communications avec le fork()



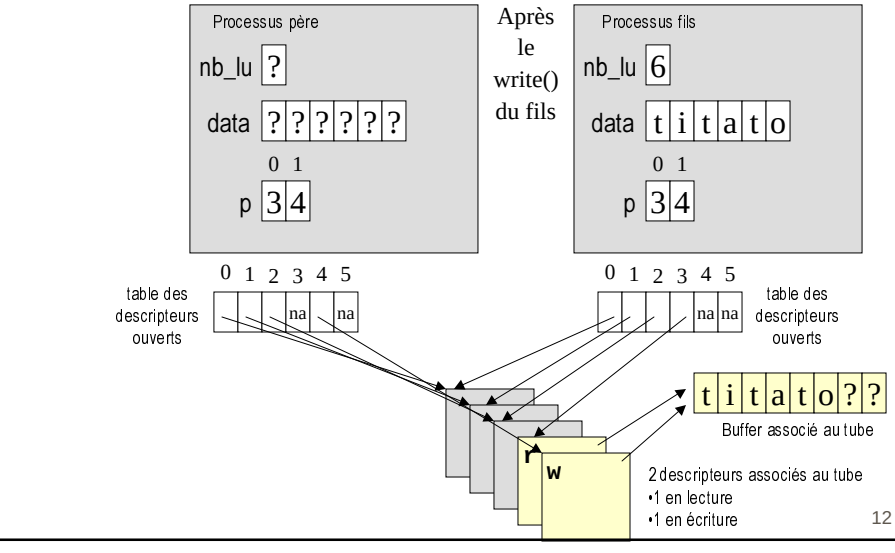
Exemple d'utilisation des tubes de communications avec le fork()



Exemple d'utilisation des tubes de communications avec le fork()

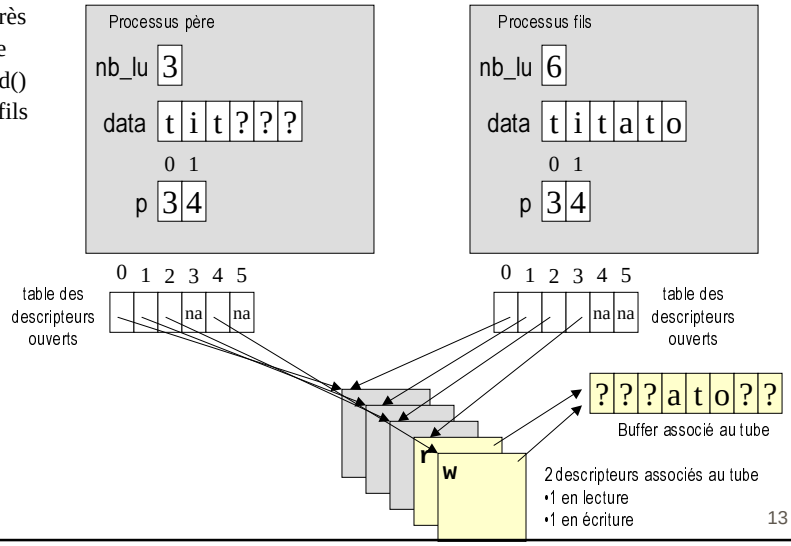


Exemple d'utilisation des tubes de communications avec le fork()



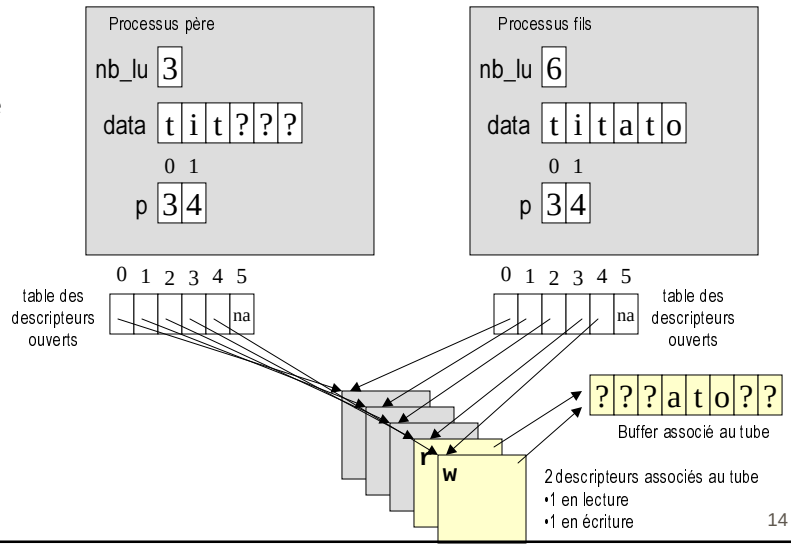
Exemple d'utilisation des tubes de communications avec le fork()

Après le read() du fils



Exemple d'utilisation des tubes de communications avec le fork()

Après le read() du père



Un programme C équivalent à la commande : ps -aux | wc -l

```
#include<stdio.h>
#define MAX 10
main()
{ int p[2];
  char data[MAX];
if (pipe(p)== -1)
        {perror("pipe");exit()};
```

```
if (fork()>0){/* le père */
   close(p[0]); /*le processus ne lit
                  jamais dans le tube */
    close(1);
    dup(p[1]); /* P[1] <=> 1 (stdout) */
 execlp("ps","ps","-aux",NULL);
} else {/* le fils */
    close(p[1]);/*le processus n 'écrit
                  jamais dans le tube */
    close(0);
    dup(p[0]);/*p[0] <=> 0 (stdin)*/
    execlp("wc","wc","-l",NULL);
  }
}
```


Les tubes nommés: FIFO

- Tubes de communications avec une référence dans le système de fichiers (visibles avec ls)
  - Des processus sans lien de parenté peuvent communiquer à l'aide des tubes nommés.
- Création d'un tube nommé dans POSIX

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

  - ref désigne le chemin d'accès au tube
  - mode désigne le droit d'accès

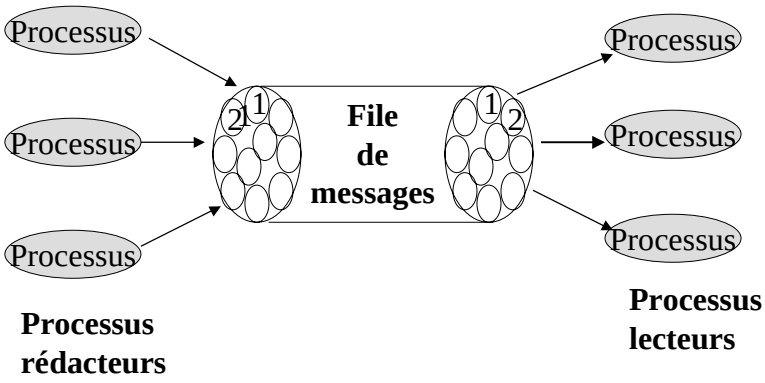
Manipulation des tubes nommés

- Les mêmes primitives système sont utilisées pour manipuler les pipes nommés :
  - open() : ouvre un FIFO selon le mode désigné
    - open bloque le processus appelant tant qu'un second processus n'a pas ouvert l'autre extrémité du FIFO
    - Après ouvertures, Les FIFOs se manipulent comme des pipes.
  - read() et write(): lire et écrire dans le FIFO
  - close() : fermeture du FIFO

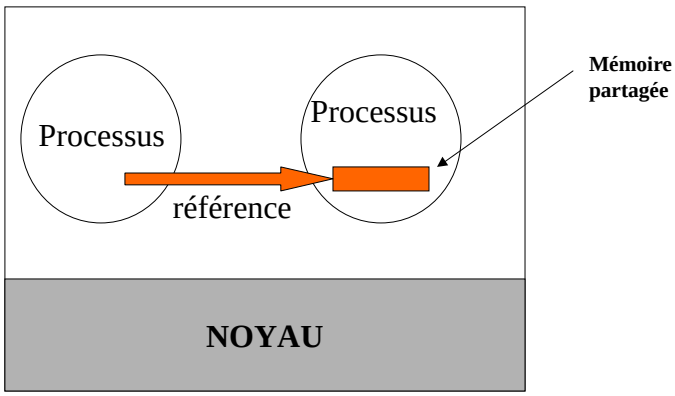
IPCs
Inter Process Communication

- En dehors du système de fichier
  - Conséquence
    - Ils ne sont pas désignés localement dans les processus par des descripteurs de fichiers
    - Impossible de rediriger les E/S standards d 'un processus sur un objet de type IPC
- Gérés par des tables
  - Les IPCs sont identifiés localement par une clé unique
    - la clé est un entier non nul
    - l 'unicité de la clé est assurée par la fonction ftock().

Les files de messages



Mémoires partagées



Les primitives des IPC System V

- Files de messages :
  - msgget : création d 'une file de message
  - msgsend() et msgrcv() permettent de lire et écrire dans une file de messages
- Mémoire partagée :
  - shmget() : créer un segment de mémoire partagée
  - shmat() : ``attacher'' un segment de mémoire partagée
  - shmdt() : ``détacher'' un segment de mémoire partagée
- Sémaphores :
  - semget() : créer un sémaphore
  - semop(): opérations sur les sémaphores

La famille des IPC System V

- IPC
  - Files de messages :
  - Mémoire partagée :
  - Sémaphores :
- Commandes shell :
  - ipcs : status des IPCs actifs
  - ipcrm : détruire un IPC (Ex : ipcrm -s msgid)

Création d 'une file de messages

- La primitive msgget():

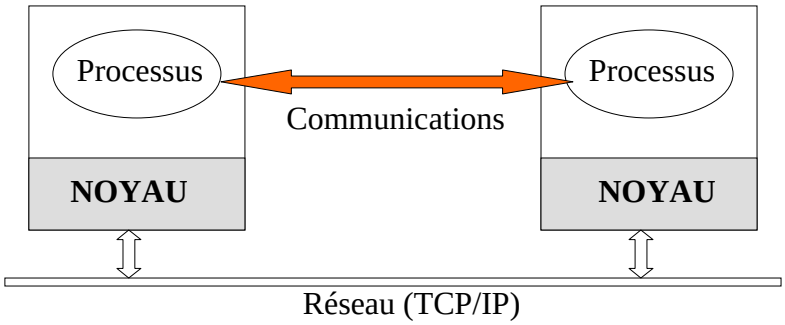
```
#include <sys/msg.h>
int msgget (key_t cle, int option)
```

  - renvoie l 'identificateur d 'une file de message ou -1 en cas d 'erreur
  - le paramètre est une combinaison des constantes IPC_CREAT, IPC_ EXCL
  - Si la clé est IPC_PRIVATE, une nouvelle file est créée

H.Bourzoufi, D. Donsez, Université de Valenciennes-ISTV, 1998-2000

Les Sockets BSD

- Mécanisme de communication entre des processus sur des noyaux séparés (Unix ou Non-Unix)
  - 2 types
    - StreamSocket (TCP/IP) DatagramSocket (UDP/IP)



Application des Sockets BSD

- Services Client-Serveurs
  - NFS, FTP, HTTP, LPD, Telnet, ...
  - Lancement de commandes à distance `rshd`
    - Sauvegarde à distance

      ***my_host*> rsh remote_host tar cf - /home | dd -o /dev/rmt0**
    - « Téléphone sur IP »

      my_host> rsh remote_host record | play