Chapitre 04 : Programmation des Tubes de Communication UNIX/Linux

Dr Mandicou BA
mandicou.ba@esp.sn
http://www.mandicouba.net

Diplôme D'Ingénieur de Conception (DIC, 2^{e} année)
Master Professionnel (1^{e} année)
Options Informatique - Systèmes Réseaux et Télécommunications

ECOLE SUPERIEURE POLYTECHNIQUE

www.esp.sn

Plan du Chapitre

Sommaire

Communication inter-processus : les tubes (pipes)

- Premier moyen d'échange de données entre processus sous UNIX
  - moyen de communication entre lecteur(s) et écrivain(s)
- Les tubes sont des objets du système de fichier
  - accès via un descripteur de fichier
  - opérations **read** et **write**
  - redirection des entrées/sorties standards depuis ou vers un tube
- Communication unidirectionnelle
  - un descripteur pour écrire (mode O_WRONLY)
  - un descripteur pour lire à l'autre bout (mode O_RDONLY)
  - pour pouvoir communiquer dans les 2 sens avec 2 processus, il faut 2 tubes
- Taille limitée finie
- L'opération de lecture y est **destructive**

Communication inter-processus : les tubes (pipes)

- Deux types de tubes
  1. **Tubes anonymes** : tubes ordinaires, non visibles dans l'arborescence, entre processus d'une même filiation fork()
  2. **Tubes nommés** : visibles dans l'arborescence, entre processus non forcément liés par filiation
- N'importe quel processus (avec les droits adéquats) peut ouvrir des tubes nommés pour communiquer
- Fichiers spéciaux gérés selon une méthodologie FIFO (First In First Out),
  - c'est-à-dire que l'ordre des caractères en entrée est conservé en sortie (premier entré, premier sortie).

Redirection vers/depuis un tube

- Motivation première des tubes, combiner
  1. connexion de la sortie standard d'un processus vers un tube
  2. connexion de l'entrée standard d'un processus depuis un tube
- Tube liant deux commandes
  - cmd1 | cmd2
- Communication d'un flot de caractères
  - un caractère lu depuis un tube est extrait du tube
  - opérations de lecture indépendantes des opérations d'écritures
    - Exemple : écrire 5 caractère, en lire 2, en écrire 3, en lire 6
- Communication en mode FIFO
  - premier caractère écrit, premier caractère lu
  - pas de possibilité de positionnement dans le tube (lseek)

Sommaire

Création

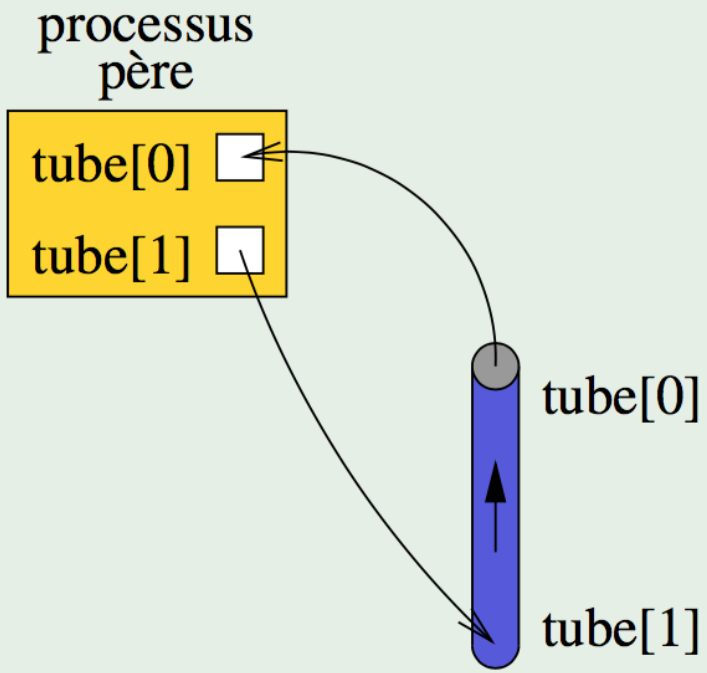
- Primitive de création d'un tube
  - int pipe(int *fd);
    - Initialise le tableau fd
      - fd : tableau de taille 2
    - fd[1] pour l'écriture
    - fd[0] pour la lecture
    - #include<unistd.h>
- Connaissance du tube
  - avoir réalisé la création
  - héritage des descripteurs

Partage d'un tube

- Un fichier sans nom
  - pas d'entrée dans le système de fichier
  - on ne peut pas utiliser open
  - création par une opération ad hoc
  - destruction automatique à la fin de l'utilisation
- Utilisation typique d'un tube
  - un processus écrivain
  - un processus lecteur
- Mécanisme de création et de partage :
  1. un tube est crée par un processus père
  2. processus père crée un processus fils
  3. le père et le fils partagent les descripteurs d'accès au tube
  4. chacun va utiliser un « bout » du tube
  5. chacun détruit le descripteur inutile

Fonctionnement : tube unidirectionnel

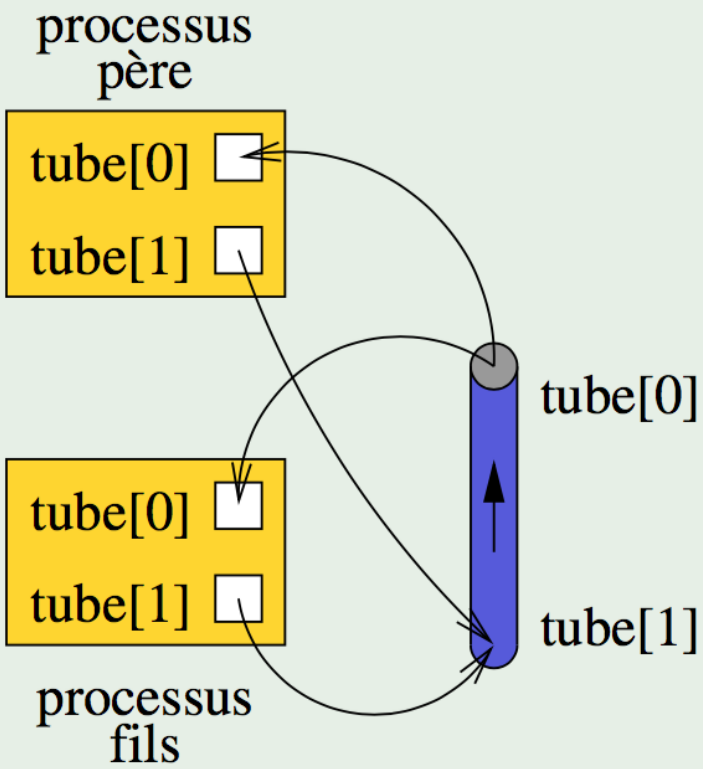
Illustration



Création du tube avec la fonction "pipe"

Fonctionnement : tube unidirectionnel

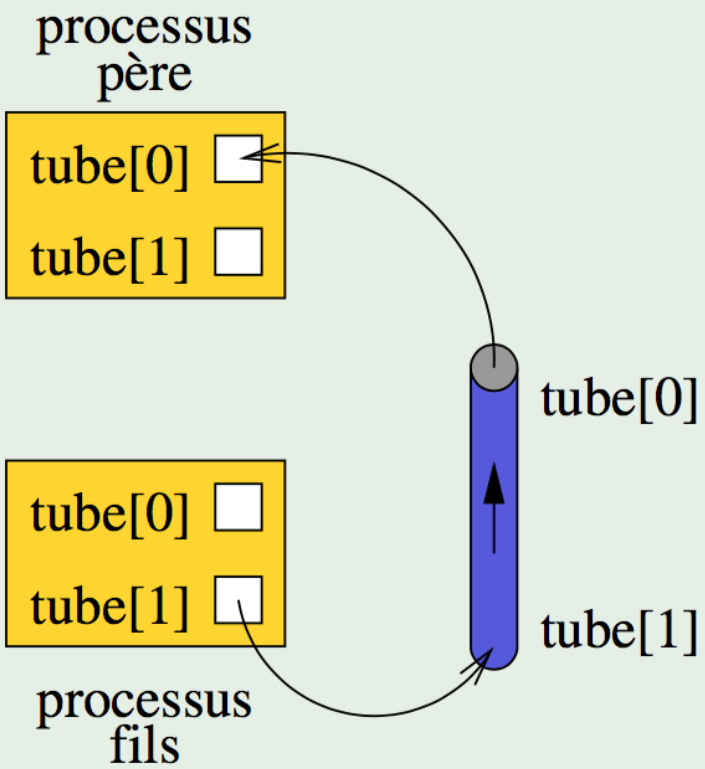
Illustration



Duplication du processus père avec la fonction "fork"

Fonctionnement : tube unidirectionnel

Illustration



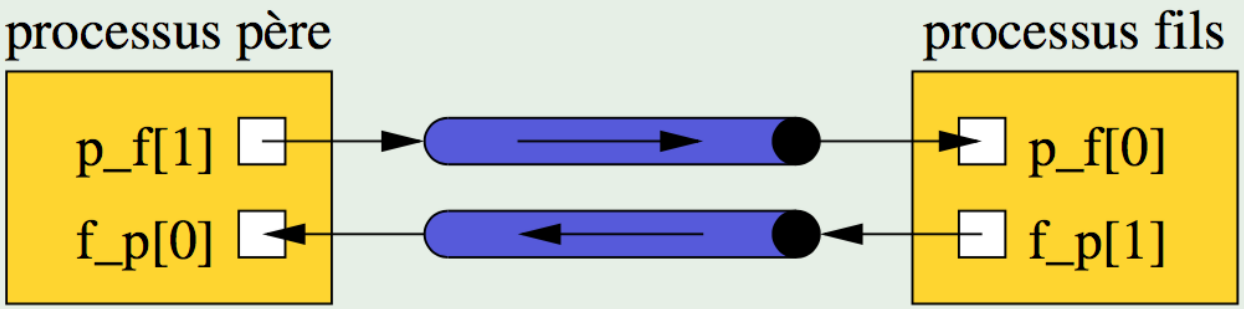
Fermeture des descripteurs inutiles avec la fonction "`close`"

Autre exemple : communication bidirectionnelle

Description

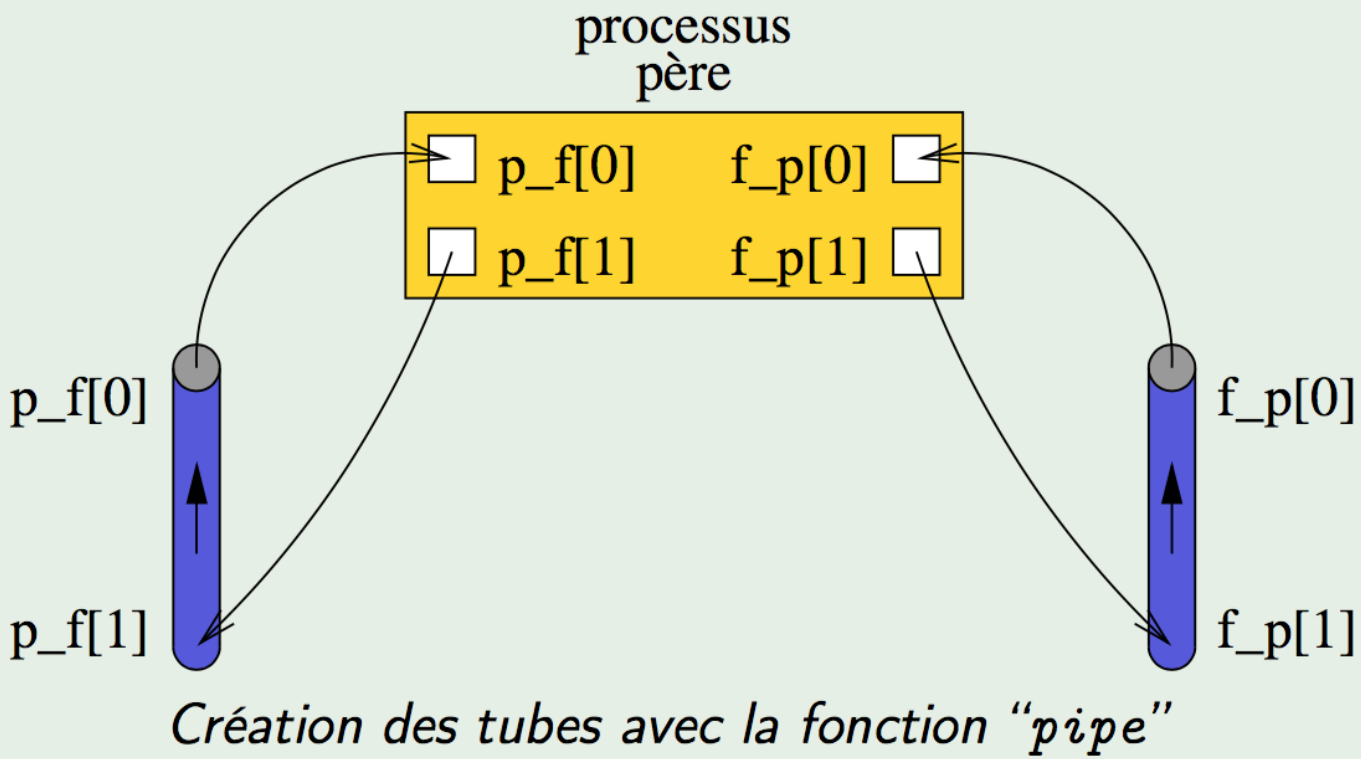
- Un tube = communication unidirectionnelle
- Comment envoyer des données dans les deux sens ?
- Solution :
  ↪ Utilisation de deux tubes !

Illustration



Fonctionnement : tube bidirectionnel

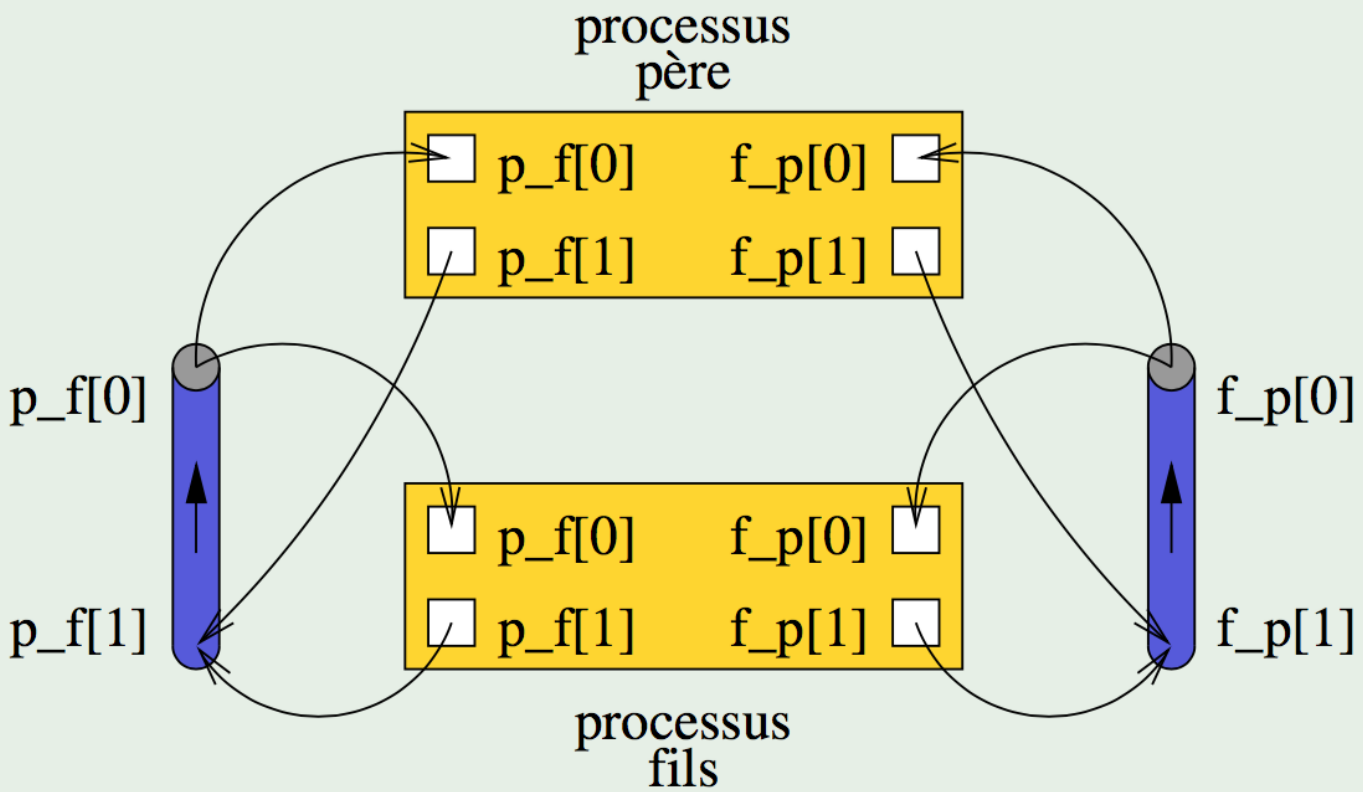
Illustration



Création des tubes avec la fonction "pipe"

Fonctionnement : tube bidirectionnel

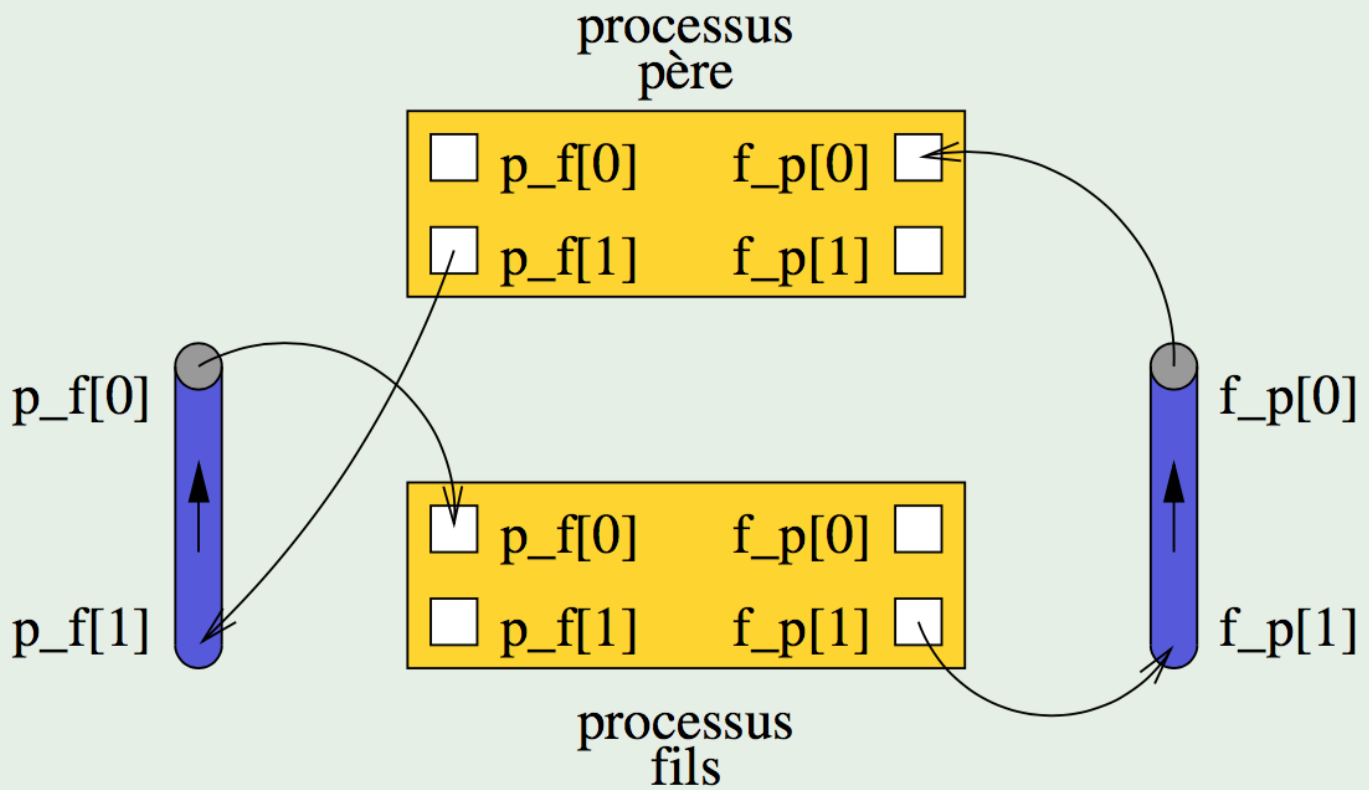
Illustration



Duplication du processus père avec la fonction "fork"

Fonctionnement : tube bidirectionnel

Illustration



Fermeture des descripteurs inutiles avec la fonction "`close`"

Caractéristiques générales des tubes

- Nombre de lecteurs
  - nombre de descripteurs associés à la lecture depuis le tube
  - s'il n'y a pas de lecteurs, le tube est inutilisable
    - Les écrivains sont prévenus : signal SIGPIPE
- Nombre d'écrivains
  1. nombre de descripteurs associés à l'écriture dans le tube
  2. s'il n'y a pas d'écrivains, le tube vide est inutilisable
     - Les lecteurs sont prévenus : notion de fin de fichier
  3. Taille bornée
     1. un tube peut être plein
     2. une écriture peut être bloquante

Lecture dans un tube

- Primitive **read**
  - ssize_t read(int fd, void *buf, size_t nbytes);
  - demande de lecture d'au plus **nbytes**
- Si le tube n'est pas vide (contient **tbytes**) :
  - read extrait **min**(tbytes, nbytes) caractères
  - écrit à l'adresse buf
  - retourne la valeur **min**(tbytes, nbytes)
- Si le tube est vide
  - Si le nombre d'écrivains est nul
    1. il n'y aura jamais plus rien à lire
    2. c'est le fin du fichier
    3. read retourne 0
  - Si le nombre d'écrivains n'est pas nul
    - processus bloqué tant qu'il n'y a pas écriture par un écrivain

Lecture dans un tube

- Réveil d'un processus bloqué en lecture sur un tube vide
  - si un écrivain réalise une écriture
    - read retourne une valeur non nulle
  - si le dernier écrivain ferme son descripteur d'accès au tube
    - explicitement par un close
    - automatiquement à sa terminaison
    - read retourne 0
- Bonne pratique
  - systématiquement fermer, au plus tôt, tous les descripteurs non utilisés :
    - garder un descripteur en écriture sur un tube peut potentiellement bloquer un processus

Écriture dans un tube

- Primitive **write**
  - ssize_t write(int fd, const void *buf, size_t nbytes);
  - demande d'écriture de nbytes caractères
- Si le nombre de lecteurs dans le tube est non nul
  - Si nbytes est inféerieur à PIPE_BUF : écriture atomique
  - sinon découpage par le système : le processus peut passer dans un état endormi
- Si le nombre de lecteurs dans le tube est nul
  - Émission du signal SIGPIPE : Comportement par défaut : terminaison

Écriture dans un tube

- Primitive **close**
  - int close(int fd)
- Fermeture d'un tube
  - Libère le descripteur
  - Pour le descripteur d'écriture, agit comme une fin de fichier
- Fermer tous les descripteurs non utilisés
  - évite les situations de blocage

Sommaire

Qu'est ce qu'un Tube nommé ?

- Appelés aussi FIFO
- Contrairement aux tubes anonymes
  - Création d'une entrée dans le système de fichiers avec un nom
  - Mais toujours pas d'opération sur le périphérique sous-jacent !
- Accessibles depuis tout processus (qui possède les droits !) :
  - Pas de filiation nécessaire
- Pour afficher les tubes : ls -l
  - Pour afficher les tubes : ls -l
  - Valeur après les droits = nombre de processus qui ont le tube ouvert

Création d'un tube nommé

- En POSIX, création par la primitive simplifié **mkfifo**
  - #include <sys/types.h>
  - #include <sys/stat.h>
  - int mkfifo(const char *filename, mode_t mode);
  - Exemple : mkfifo("myfifo", 0666); // octal 0666 = permissions lecture + écriture
- Crée un tube nommé de nom filename et de permissions mode
  - Retourne 0 en cas de succès, -1 en cas d'échec
  - Une fois crée, n'importe quel processus en ayant les droits peut ouvrir le tube nommé comme il le ferai avec un fichier ordinaire pour lecture et écriture.

Ouverture d'un tube nommé

- Ouvertures possibles par les processus connaissant le nom du tube
  - #include <sys/types.h> et #include <sys/stat.h>
  - int open(const char *filename, int flags);
  - Exemples :
    1. open("fifo", O_RDONLY); // ouverture en lecture seule
    2. open("fifo", O_WRONLY); // ouverture en écriture seule
    3. open("fifo", O_RDONLY | O_NONBLOCK); // Réussit immédiatement
    4. open("fifo", O_WRONLY | O_NONBLOCK); // Réussit ou échoue
- Avant de lire ou d'écrire, chaque extrémité du tube doit être ouverte :
  - flags spécifie les droits d'ouverture (lecture, écriture, ou les 2)
  - par défaut, l'ouverture du tube en lecture est bloquante tant qu'il n'y a pas ouverture en écriture et vice versa (le producteur et le consommateur sont alors synchronisés)
  - Avec le flag O_NONBLOCK (mode non bloquant), open se comporte de façon asymétrique

Ouverture d'un tube nommé

- En lecture seule, open retourne :
  - le descripteur immédiatement sans attendre l'ouverture en écriture par le producteur
- En écriture seule, open retourne:
  - le descripteur immédiatement ou échoue (errno = ENXIO) :
    1. si il n'est pas ouvert en lecture en mode non bloquant par le producteur,
    2. ou si le lecteur est bloqué sur l'ouverture de celui-ci en mode bloquant
- Pourquoi ce comportement ? Ceci pour éviter que :
  1. le processus qui vient d'ouvrir le tube nommé, n'écrive dans le tube avant qu'il n'y ait de lecteur (qu'un processus l'ait ouvert en lecture)
  2. et ce qui engendrerait un signal SIGPIPE (tube détruit), ce qui n'est pas vrai car le tube n'a pas encore été utilisé
- Toutes les opérations de lecture écriture qui suivent sont alors non bloquantes

Opération de LECTURE-ÉCRITURE

- Peuvent être lu ou écris comme des fichiers ordinaires :
  - en utilisant les primitives systèmes (bas niveau) : read, write
  - en utilisant les fonctions de la librairie (haut niveau)
- Exemples :
  - FILE *fout= open(tube ,"w");
  - fprintf(fout, "Coucou le monde");
- Pour les tubes ouverts en mode bloquant (mode par défaut):
  - la lecture depuis un tube vide est bloquante (attente de données)
  - l'écriture dans un tube plein est bloquante (attente d'une possibilité d'écriture)
- Taille du buffer du tube : PIPE_BUF peut être définit dans <limits.h>
  - PIPE_BUF = 4096 (LINUX)
  - PIPE_BUF = 512 minimum si le système est conforme POSIX
  - Cette taille est la taille maximum qui garantie l'atomicité de l'écriture

Opération de LECTURE

☛ nb_lu = read(fd, buffer, TAILLE_READ);

1. Si le tube n'est pas vide et contient **taille** caractères
   - Lecture de nb_lu = min (**taille**, **TAILLE_READ**) caractères.
2. Si le tube est vide
   - Si le nombre d'écrivains est nul
     - alors c'est la fin de fichier et nb_lu est nul
   - Si le nombre d'écrivains est non nul
     - si lecture bloquante alors sommeil
     - si lecture non bloquante alors en fonction de l'indicateur

       O_NONBLOCK : nb_lu= -1 et errno=EAGAIN.

       O_NDELAY : nb_lu = 0.

Opération d'écriture

- nb_ecrit = write(fd, buf, n);
- L'écriture est atomique si le nombre de caractères à écrire est inférieur à PIPE_BUF, la taille du tube sur le système. (cf <limits.h>).
- Si le nombre de lecteurs est nul
  - Envoi du signal SIGPIPE à l'écrivain et errno = EPIPE.
- Sinon
  1. Si l'écriture est bloquante, il n'y a retour que quand
     - les n caractères ont été écrits dans le tube.
  2. Si écriture non bloquante
     - Si n > PIPE_BUF, retour avec un nombre inférieur a n éventuellement -1 !
  3. Si n = PIPE_BUF
     - Et si n emplacements libres, écriture nb_ecrit = n
  4. Sinon retour -1 ou 0.

Attention !

- ☛ L'appel à read retourne toujours le maximum de données possibles,
  - même si les appels à write sont issus de processus différents
- ☛ Il est donc nécessaire que les paquets envoyés soit toujours de la même taille.
- ☛ Quand on écrit ou lit dans/depuis un tube nommé, read() retournera EOF quand tous les producteurs seront fermés
- ☛ et write() enverra le signal SIGPIPE quand il n'y aura plus de consommateur.
- ☛ Si SIGPIPE est bloqué ou ignoré, on peut tester errno = EPIPE.

Suppression du Tube

- ☛ Suppression du tube si et seulement si elle est explicitement demandée et pas d'ouverture en cours
- ☛ Si suppression alors qu'ils existent des lecteurs et écrivain, fonctionnement comme un fichier ordinaire
  - Un nœud est supprimé quand:
    - le nombre de liens physiques est nul.
      - Fonction **unlik** ou la commande **rm**.

        int unlink(const char *pathname);

        Renvoie 0 succès ou -1 échec
    - Le nombre de liens internes est nul
      - Nombres de lecteurs et écrivains sont nuls
  - Si nombre de lien physique est nul, mais le nombre de lecteurs et/ou écrivains est non nul
    - Tube nommé devient un tube anonyme.

Chapitre 04 : Programmation des Tubes de Communication UNIX/Linux

Dr Mandicou BA
mandicou.ba@esp.sn
http://www.mandicouba.net

Diplôme D'Ingénieur de Conception (DIC, 2^e année)
Master Professionnel (1^e année)
Options Informatique - Systèmes Réseaux et Télécommunications

ECOLE SUPERIEURE POLYTECHNIQUE

www.esp.sn