Outils de communication entre processus Linux

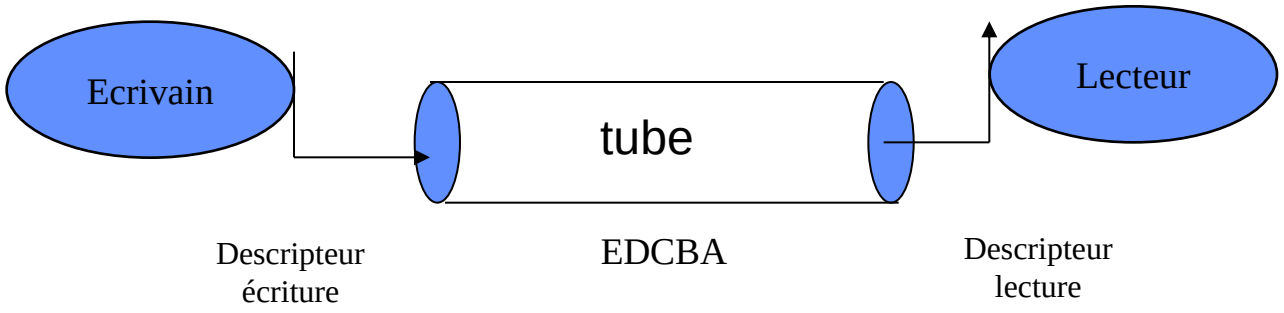
Deux familles

- **Les outils gérés avec le SGF : les tubes et les sockets**
- **Les outils IPC gérés dans des tables du systèmes et repérés par une clef : les MSQ, les segments de mémoire partagée, les sémaphores**

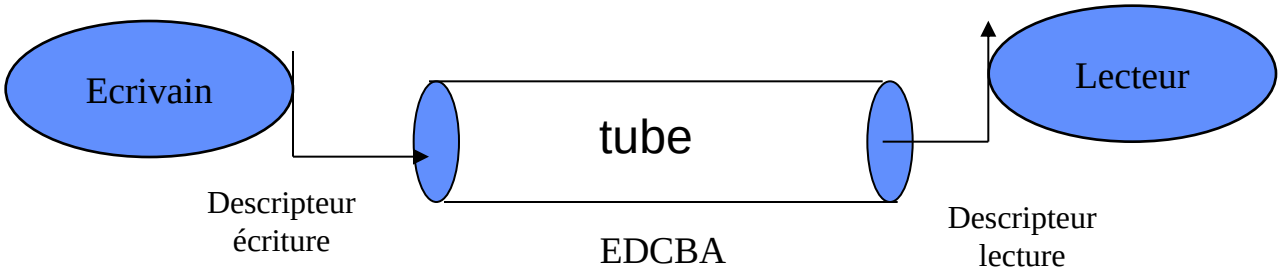
Outils de communication entre processus Linux

LES TUBES

- **Outils gérés avec le SGF, accessibles via des descripteurs**
- **Communication unidirectionnelle entre processus, en mode FIFO**
- **Tubes anonymes : entre processus d'une même famille**
- **Tubes nommés : entre n'importe quel processus**



Les tubes : les tubes anonymes



LES TUBES Anonymes

- **Fichier sans nom, seulement accessible par les deux descripteurs**
- **Seuls les processus ayant connaissance par héritage des descripteurs peuvent utiliser le tube → processus créateur et ses descendants**

Les tubes : les tubes anonymes
CREATION

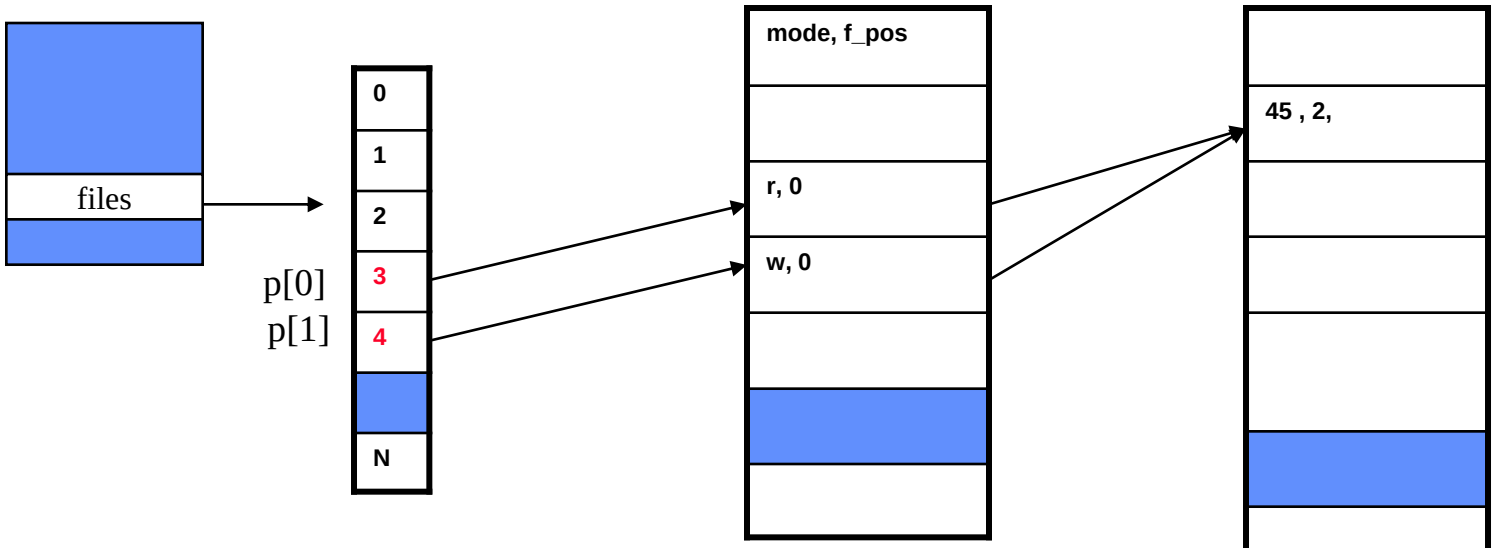
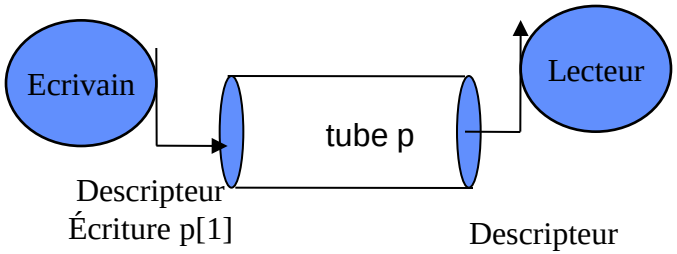


Table des fichiers ouverts
Du processus

Table des fichiers ouverts
Mode : lecteur, écriture…

f_pos : pointeur de fichier ; octet courant

Table des inodes mémoires
f_count : nombre de référence
au même fichier




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

Les tubes : les tubes anonymes

Exemple de communication entre processus par l'intermédiaire d'un tube. Un processus fils écrit à destination de son père la chaîne de caractères « bonjour ».

1. Le processus père ouvre un tube en utilisant la fonction pipe().

2. Le processus père crée un fils en utilisant la fonction fork().

3. Les descripteurs en lecture et écriture du tube sont utilisables par les 2 processus. Chacun des deux processus ferme le descripteur qui lui est inutile : ainsi, le processus père ferme le descripteur en écriture et le processus fils ferme le descripteur en lecture.

4. Le fils envoie un message à son père.

5. Le père lit le message.

```
/*****************************************************************/
/*    Communication unidirectionnelle entre un père et son fils    */
/*****************************************************************/
main () {
int p[2], pid_t retour;
char chaine[7];
pipe(p);
retour = fork();
if (retour == 0)
{ /* le fils écrit dans le tube */
      close (p[0]); /* pas de lecture sur le tube */
      write (p[1], "bonjour", 7);
      close (p[1]); exit(0); }
else
{ /* le père lit le tube */
      close (p[1]); /* pas d'écriture sur le tube */
      read(p[0], chaine, 7);
      close (p[0]);
      wait;
}
```

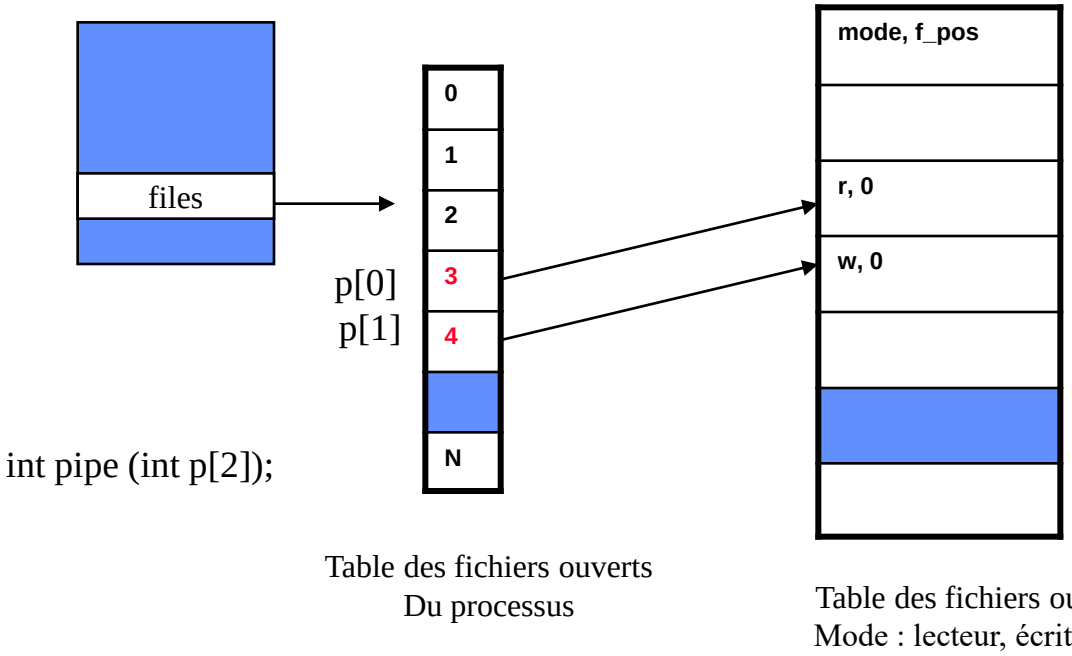
Les tubes : les tubes anonymes

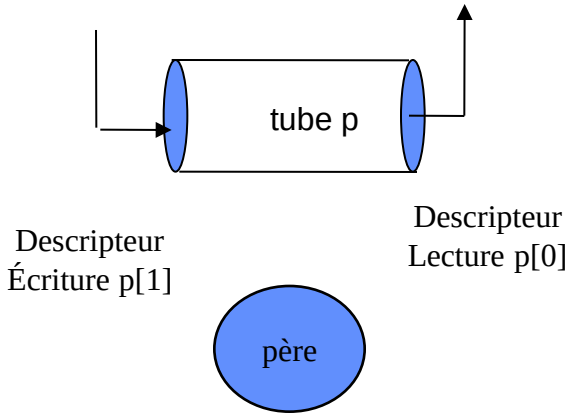
Exemple de communication entre processus par l'intermédiaire d'un tube. Un processus fils écrit à destination de son père la chaîne de caractères « bonjour ».

1. Le processus père ouvre un tube en utilisant la fonction pipe().

2. Le processus père crée un fils en utilisant la fonction fork().

3. Les descripteurs en lecture et écriture du tube sont utilisables par les 2 processus. Chacun des deux processus ferme le descripteur qui lui est inutile : ainsi, le processus père ferme le descripteur en écriture et le processus fils ferme le descripteur en lecture.





f_pos : pointeur de fichier ; octet courant

Les tubes : les tubes anonymes

Exemple de communication entre processus par l'intermédiaire d'un tube. Un processus fils écrit à destination de son père la chaîne de caractères « bonjour ».

1. Le processus père ouvre un tube en utilisant la fonction pipe().

2. Le processus père crée un fils en utilisant la fonction fork().

3. Les descripteurs en lecture et écriture du tube sont utilisables par les 2 processus. Chacun des deux processus ferme le descripteur qui lui est inutile : ainsi, le processus père ferme le descripteur en écriture et le processus fils ferme le descripteur en lecture.

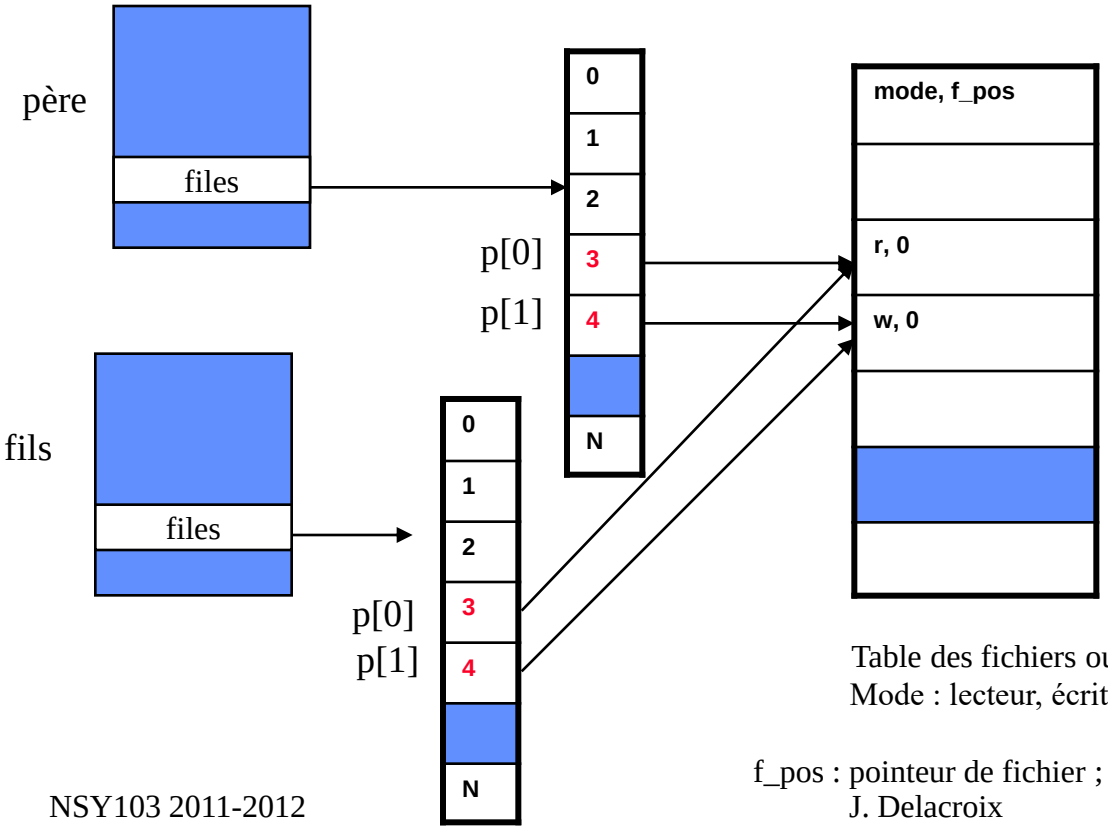
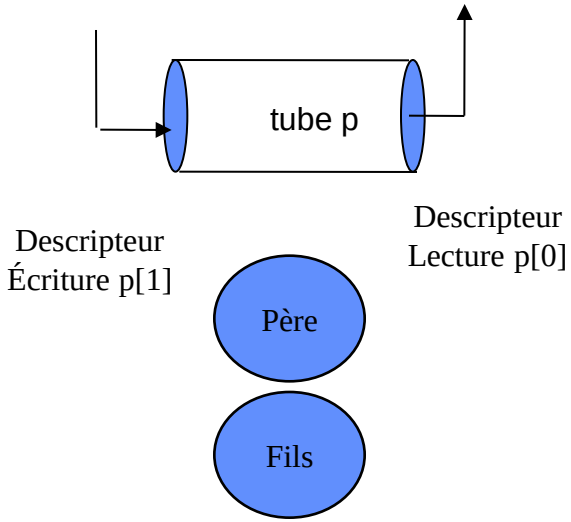


Table des fichiers ouverts
Mode : lecteur, écriture…

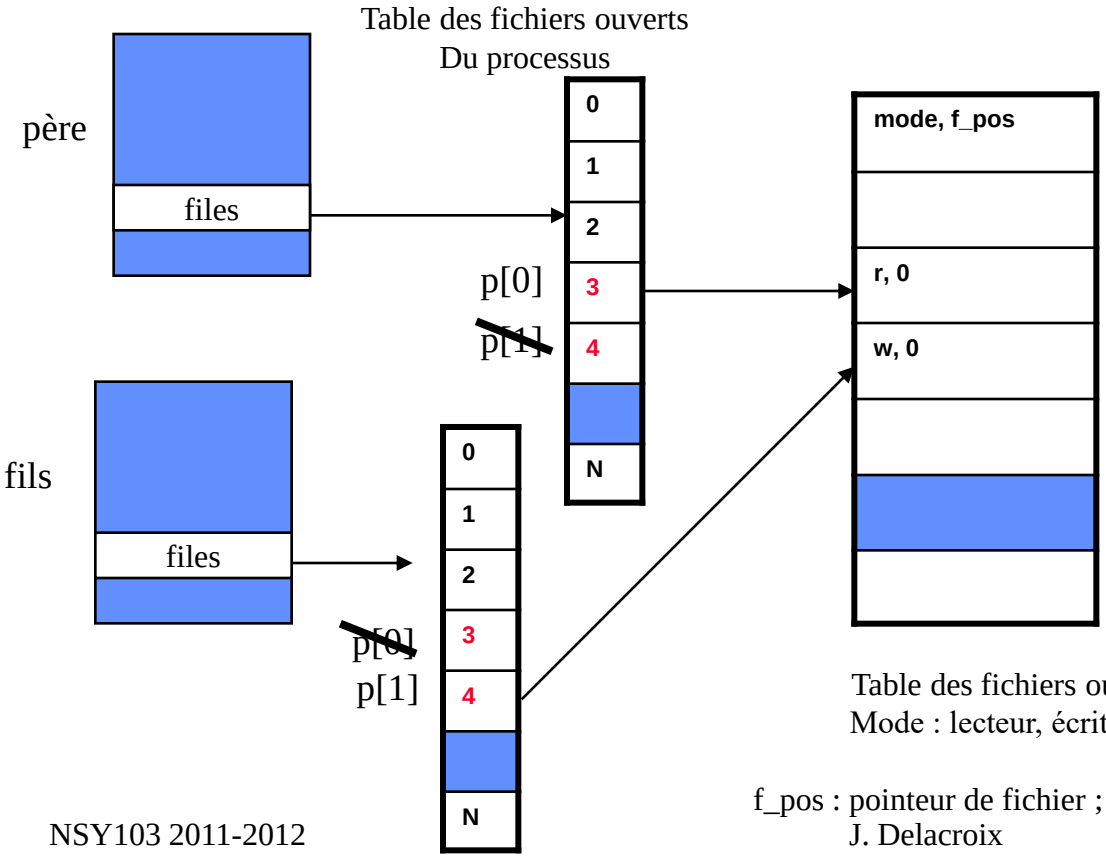
f_pos : pointeur de fichier ; octet courant

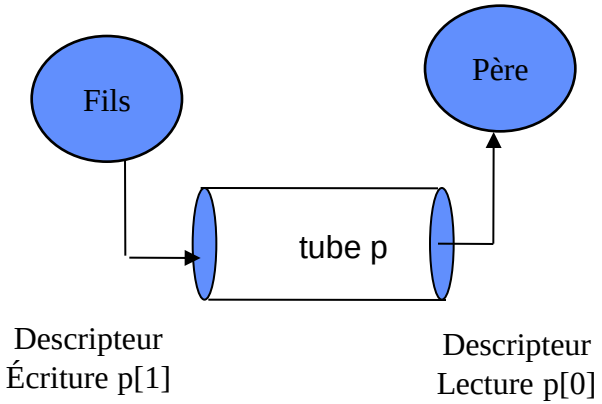


Les tubes : les tubes anonymes

Exemple de communication entre processus par l'intermédiaire d'un tube. Un processus fils écrit à destination de son père la chaîne de caractères « bonjour ».

1. **Le processus père ouvre un tube en utilisant la fonction pipe().**
2. **Le processus père crée un fils en utilisant la fonction fork().**
3. **Les descripteurs en lecture et écriture du tube sont utilisables par les 2 processus. Chacun des deux processus ferme le descripteur qui lui est inutile : ainsi, le processus père ferme le descripteur en écriture et le processus fils ferme le descripteur en lecture.**





f_pos : pointeur de fichier ; octet courant

Les tubes : les tubes anonymes
PRIMITIVES

Fermeture de descripteur

Un processus ferme un descripteur de tube fd en utilisant la primitive close() :

int close(int fd);

Le nombre de descripteurs ouverts en lecture détermine le nombre de lecteurs existants pour le tube.

Le nombre de descripteurs ouverts en écriture détermine le nombre d'écrivains existants pour le tube.

Un descripteur fermé ne permet plus d'accéder au tube et ne peut pas être régénéré.

Les tubes : les tubes anonymes
PRIMITIVES

Lecture

La lecture s'effectue par le biais de la primitive read()

int read(int desc[0], char *buf, int nb);

La primitive permet la lecture de nb caractères depuis le tube desc, qui sont placés dans le tampon buf. Elle retourne en résultat le nombre de caractères réellement lus.

L'opération de lecture répond à la sémantique suivante :

- **si le tube n'est pas vide et contient taille caractères, la primitive extrait du tube min(taille, nb) caractères qui sont lus et placés à l'adresse buf ;**
- **si le tube est vide et que le nombre d'écrivains est non nul, la lecture est bloquante. Le processus est mis en sommeil jusqu'à ce que le tube ne soit plus vide ;**
- **si le tube est vide et que le nombre d'écrivains est nul, la fin de fichier est atteinte. Le nombre de caractères rendu est nul.**

Les tubes : les tubes anonymes
PRIMITIVES

Ecriture

L'écriture dans un tube anonyme s'effectue par le biais de la primitive write()

int write(int desc[1], char *buf, int nb);

La primitive permet l'écriture de nb caractères placés dans le tampon buf dans le tube desc. Elle retourne en résultat le nombre de caractères réellement écrits.

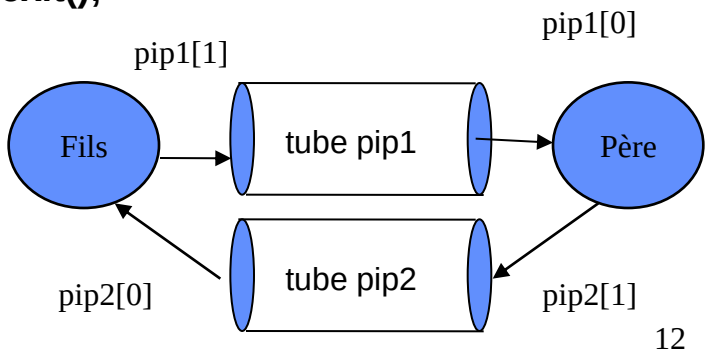
L'opération d'écriture répond à la sémantique suivante :

- **si le nombre de lecteurs dans le tube est nul, alors une erreur est générée et le signal SIGPIPE est délivré au processus écrivain, et le processus se termine. L'interpréteur de commandes shell affiche par défaut le message « Broken pipe » ;**

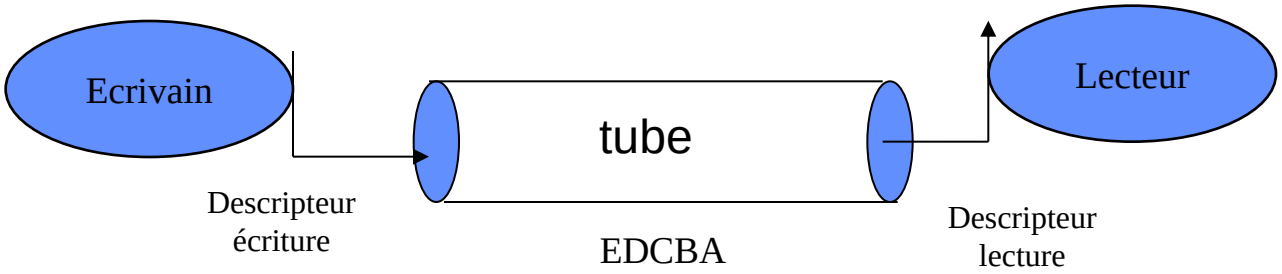
- **si le nombre de lecteurs dans le tube est non nul, l'opération d'écriture est bloquante jusqu'à ce que les nb caractères aient effectivement été écrits dans le tube.**

Les tubes : les tubes anonymes
Communication bidirectionnelle

```
/*****************************************************************/
/*    Communication bidirectionnelle entre un père et son fils    */
/*****************************************************************/
#include <stdio.h>
int pip1[2];       /*  descripteurs pipe 1  */
int pip2[2];       /*  descripteurs pipe 2  */
main()
{
int idfils;
char rep[7], mesg[5];
/* ouverture tubes */
 (pipe(pip1))
 (pipe(pip2))
/* création processus */
idfils=fork()
if(idfils) {
        /*le premier tube sert dans le sens père vers fils
          il est fermé en lecture */
       close(pip1[0]);
       /*le second tube sert dans le sens fils vers père
          il est fermé en écriture*/
       close(pip2[1]);
       /* on envoie un message au fils par le tube 1*/
         write(pip1[1],"hello",5)
       /* on attend la réponse du fils par le tube 2
          */
         iread(pip2[0],rep,7);
         printf("message du fils: %s\n",rep);
         wait; }
else   {
         /*fermeture du tube 1 en écriture */
         close(pip1[1]);
         /* fermeture du tube 2 en lecture */
         close(pip2[0]);
         /* attente d'un message du père */
         read(pip1[0],mesg,5)
         printf("la chaine reçue par le fils est:
       %s\n",mesg);
         /* envoi d'un message au père */
         write(pip2[1],"bonjour",7)
         exit();
       }
}
```



Les tubes : les tubes nommés



LES TUBES Nommés

- **Fichier avec nom, seulement accessible par les deux descripteurs**
- **Tous les processus ayant connaissance des descripteurs peuvent utiliser le tube → pas d'obligation de filiation**

Les tubes : les tubes nommés
PRIMITIVES

Création et ouverture

Un tube nommé est créé par l'intermédiaire de la fonction mkfifo()

int mkfifo (const char *nom, mode_t mode);

Le paramètre nom correspond au chemin d'accès dans l'arborescence de fichiers pour le tube.

Le paramètre mode correspond aux droits d'accès associés au tube

L'ouverture d'un tube nommé s'effectue en utilisant la primitive open()

int open (const char *nom, int mode_ouverture);

La primitive renvoie un seul descripteur correspond au mode d'ouverture spécifié (lecture seule, écriture seule, lecture/écriture).

La primitive open() appliquée au tube nommé est bloquante.

- **demande d'ouverture en lecture est bloquante tant qu'il n'existe pas d'écrivain sur le tube.**
- **demande d'ouverture en écriture est bloquante tant qu'il n'existe pas de lecteur sur le tube.**

Ce mécanisme permet à deux processus de se synchroniser et d'établir un rendez-vous en un point particulier de leur exécution.

Les tubes : les tubes nommés
PRIMITIVES

Fermeture de descripteur

Un processus ferme un descripteur de tube fd en utilisant la primitive close() :

```
int close(int fd);
```

Lecture

La lecture s'effectue par le biais de la primitive read()

```
int read (int desc, char *buf, int nb);
```

Ecriture

L'écriture s'effectue par le biais de la primitive write()

```
int write (int desc, char *buf, int nb);
```

Les tubes : les tubes nommés

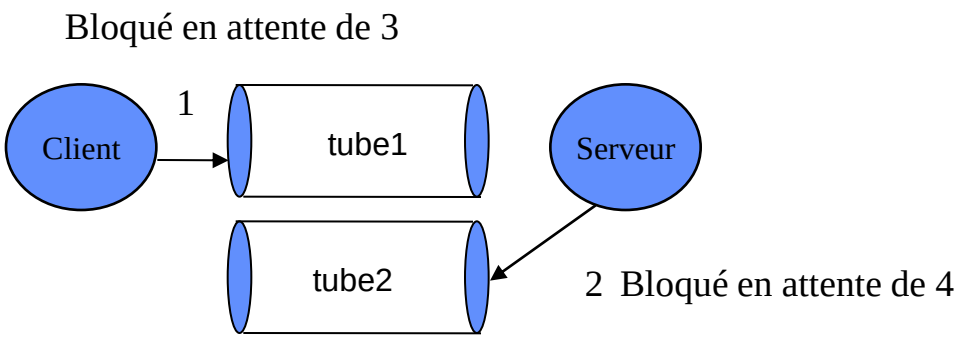
```
/************************************************/
/* Processus lecteur sur le tube nommé  */
/************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
main ()
{
char zone[11];
int tub;
/* ouverture du tube */
tub = open ("fictub", O_RDONLY);
/* lecture dans le tube */
read (tub, zone, 10);
printf ("processus lecteur du tube fictub: j'ai
      lu %s", zone);
/* fermeture du tube */
close(tub);
}
```

```
/************************************************/
/*   Processus écrivain sur le tube nommé */
/************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
main ()
{
mode_t mode;
int tub;
mode = S_IRUSR | S_IWUSR;
/*création du tube nommé */
mkfifo ("fictub", mode);
/* ouverture du tube */
tub = open ("fictub", O_WRONLY);
/* écriture dans le tube */
write (tub, "0123456789", 10);
/* fermeture du tube */
close(tub);
```

Les tubes : les tubes nommés. Exemple d'interblocage

```
   CLIENT
   /* ouverture du tube tube1 en écriture  */
1  tub1 = open ("tube1", O_WRONLY);  -- en attente de l'ouverture en lecture de tube1
   /* ouverture du tube tube2 en lecture */
4  tub2 = open ("tube2", O_RDONLY);

   SERVEUR
2  /*ouverture du tube 2 en écriture */
   tub2 = open ("tube2", O_WRONLY);  -- en attente de l'ouverture en lecture de tube2
3  /* ouverture du tube 1 en lecture */
   tub1 = open ("tube1", O_RDONLY);
```



LINUX : processus et outils de communication
Partie 2

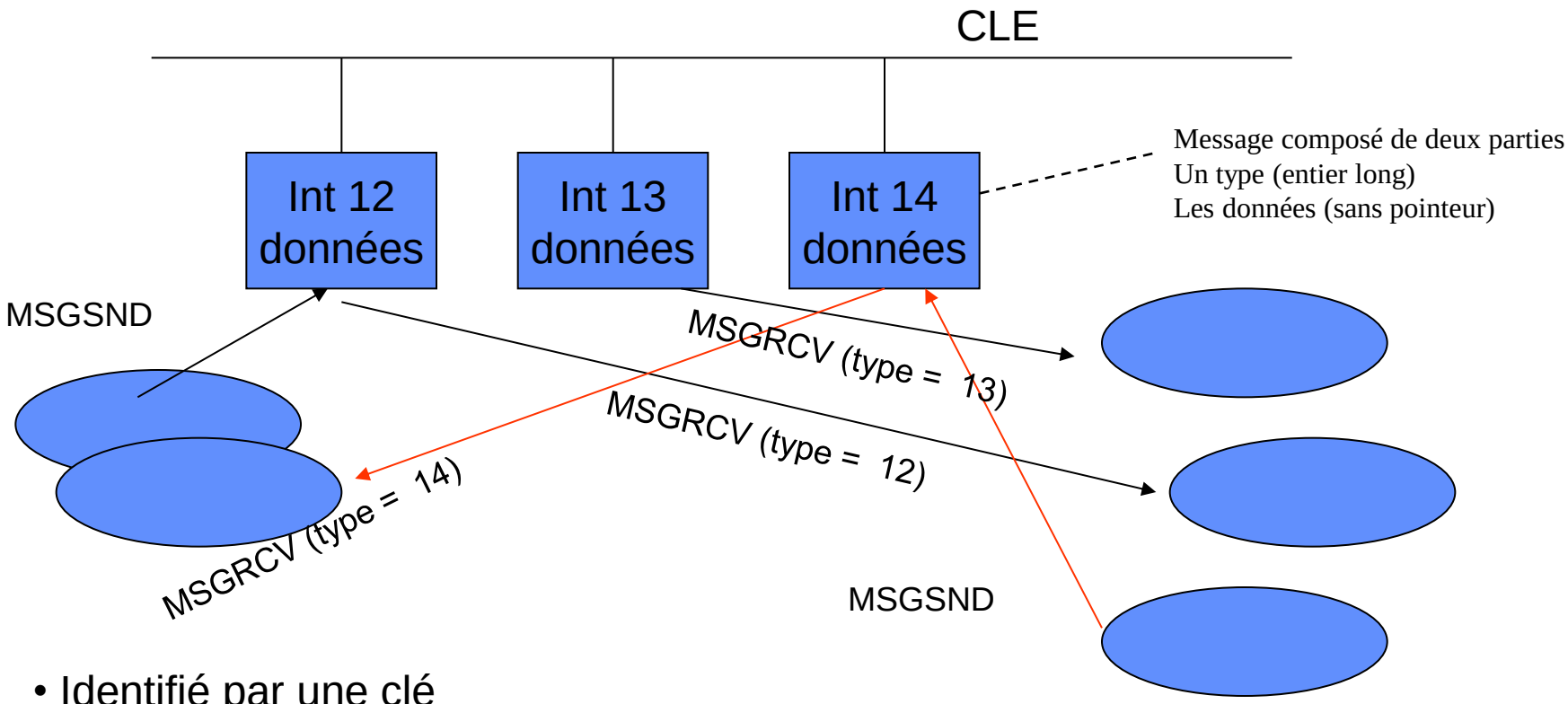
Outils IPC (*Inter Process Communication*)

- **Outils de communication n'appartenant pas au système de gestion de fichiers.**
  - **les files de messages ou MSQ (*Messages Queues*) ;**
  - **les régions de mémoire partagée ;**
  - **les sémaphores.**

Outils IPC
Caractéristiques communes

- **Gérés dans des tables du système, une par outils.**
- **identifié de manière unique par un identifiant externe appelé la clé (qui a le même rôle que le chemin d'accès d'un fichier) et par un identifiant interne (qui joue le rôle de descripteur).**
- **Accessible à tout processus connaissant l'identifiant interne de cet outil, soit par héritage ou par une demande explicite au système au cours de laquelle le processus fournit l'identifiant externe de l'outil IPC.**
- **La clé est une valeur numérique de type key_t. Les processus désirant utiliser un même outil IPC pour communiquer doivent se mettre d'accord sur la valeur de la clé référençant l'outil. Ceci peut être fait de deux manières :**
  - **la valeur de la clé est figée dans le code de chacun des processus ;**
  - **la valeur de la clé est calculée par le système à partir d'une référence commune à tous les processus. Cette référence est composée de deux parties, un nom de fichier et un entier. Le calcul de la valeur de la clé à partir cette référence est effectuée par la fonction ftok(), dont le prototype est :**
- **#include <sys/ipc.h>**
- **key_t ftok (const char *ref, int numero);**

Outils IPC
Les files de message ou MSQ



- Identifié par une clé
- Entre processus quelconque connaissant le clé
- Bidirectionnel
- Multiplexage

Exemple Files de messages

PROCESSUS A

```
#include  <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define cle 17 /* indentifiant externe */
struct msgbuf_exo {long mtype;
                   char mtext[20]; }
struct msgbuf_exo msgp;
main ()  {
int msqid ; /* identifiant interne de la MSQ */

/* allocation de la msq */
msqid = msgget (cle, IPC_CREAT | IPC_EXCL |
      0666);

/* ecriture message dans la msq */
msgp.mtype  =12;   on associe un type au
      message

strcpy(msgp.mtext, "ceci est un message");

msgsnd (msqid, &msgp, strlen(msgp.mtext), 0);
}
```

PROCESSUS B

```
#include  <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define cle 17/* indentifiant externe */
struct msgbuf_exo {long mtype;
                   char mtext[20]; }
struct msgbuf_exo msgp;
main ()  {
int msqid ; /* identifiant interne de la MSQ */

/* recuperation de la msq */
msqid = msgget (cle, 0);

/* lecture message dans la msq de type 12*/
msgrcv (msqid, &msgp, 19, 12, 0);

/* destruction de la msq */
msgctl (msqid, IPC_RMID, NULL);
}
```

Outils IPC
Les files de message ou MSQ

- **L'accès à une file de message s'effectue par l'intermédiaire de la primitive msgget(). Cette primitive permet :**
  - **la création d'une nouvelle file de messages ;**
  - **l'accès à une file de messages déjà existante.**

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

combinaison des constantes IPC_CREAT, IPC_EXCL et de droits d'accès

Clenv = ftok ("/home/delacroix/essai.c", 24)

msqid = msgget (cleenv, IPC_CREAT | IPC_EXCL | 0666); : création d'un file nouvelle de cle « clenv » avec des accès en lecture/écriture pour tous

msqid = msgget (cleenv, IPC_CREAT | 0660); : création ou accès à une file existante de cle « clenv » avec des accès en lecture/écriture pour le propriétaire et le groupe du propriétaire

msqid = msgget (cleenv, 0); : accès à une file existante de cle « clenv »

Exemple Files de messages

PROCESSUS A

```
#include  <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define cle 17 /* indentifiant externe */
struct msgbuf_exo {long mtype;
                   char mtext[20]; }
struct msgbuf_exo msgp;
main ()  {
int msqid ; /* identifiant interne de la MSQ */

/* allocation de la msq */
msqid = msgget (cle, IPC_CREAT | IPC_EXCL |
      0666);

/* ecriture message dans la msq */
msgp.mtype  =12;   on associe un type au
      message

strcpy(msgp.mtext, "ceci est un message");

msgsnd (msqid, &msgp, strlen(msgp.mtext), 0);
}
```

PROCESSUS B

```
#include  <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define cle 17/* indentifiant externe */
struct msgbuf_exo {long mtype;
                   char mtext[20]; }
struct msgbuf_exo msgp;
main ()  {
int msqid ; /* identifiant interne de la MSQ */

/* recuperation de la msq */
msqid = msgget (cle, 0);

/* lecture message dans la msq de type 12*/
msgrcv (msqid, &msgp, 19, 12, 0);

/* destruction de la msq */
msgctl (msqid, IPC_RMID, NULL);
}
```

Outils IPC
Les files de message ou MSQ

- **La communication au travers d'une file de messages peut être bidirectionnelle.**

- **Chaque message comporte les données en elles-mêmes ainsi qu'un type qui permet de faire du multiplexage dans la file de messages et de désigner le destinataire d'un message.**

Format des messages
Un message est toujours composé de deux parties :

```
struct message {
        long mtype;
        int n1;
        char[4];
        float fl1; };
```

la première partie constitue le type du message. C'est un entier long positif ;

la seconde partie est composée des données proprement dites.

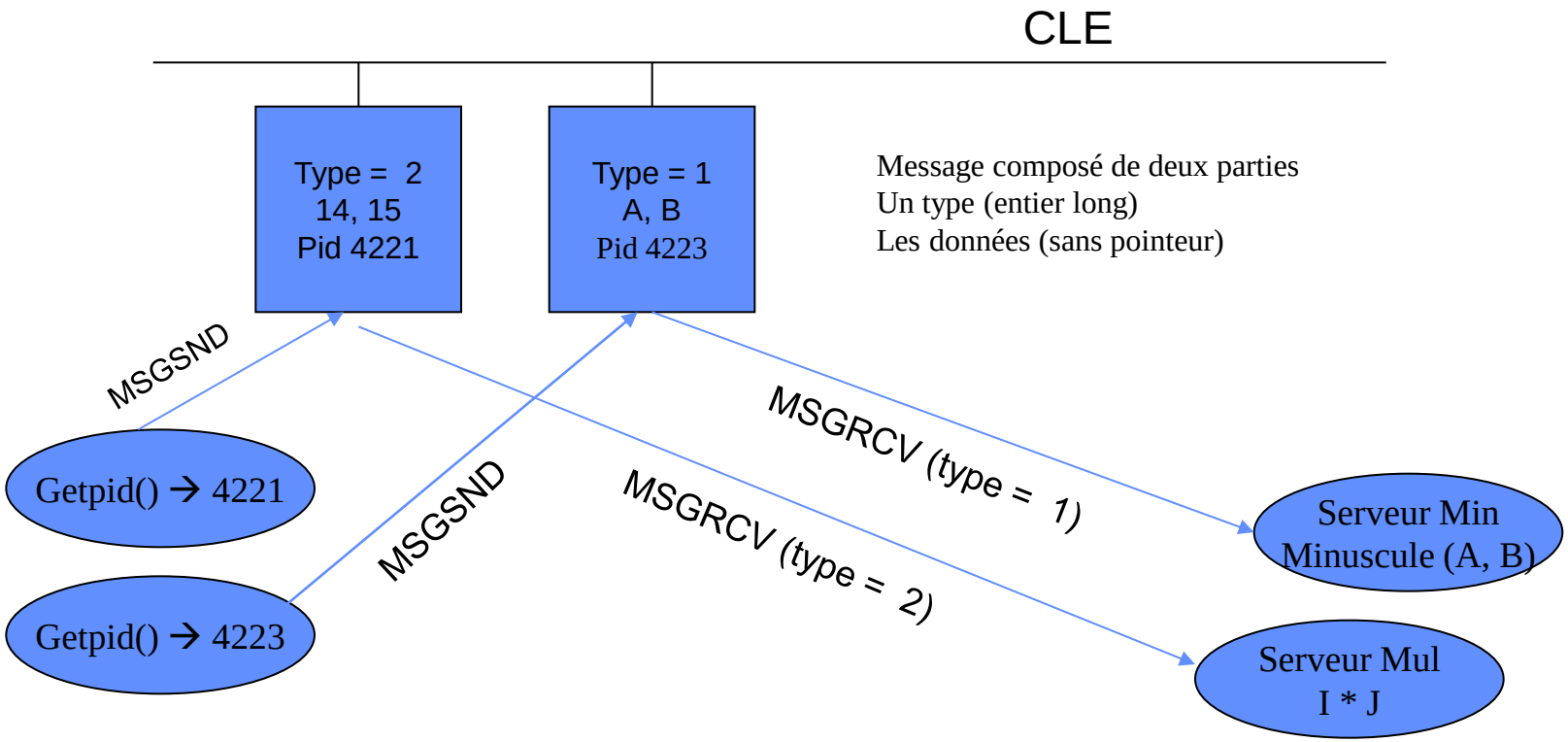
- Si letype==0 le premier message de n'importe quel type est extrait
- Si letype > 0 le premier message de type== letype est extrait
- Si letype < 0 le premier message de type < letype est extrait

- **L'envoi et la réception d'un message s'effectue via les primitives :**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd (int idint, const void *msg, int longueur, int option);
int msgrcv (int idint, const void *msg, int longueur, long letype, int option);
```

Identifiant clé — message — taille du message en données — type à prélever

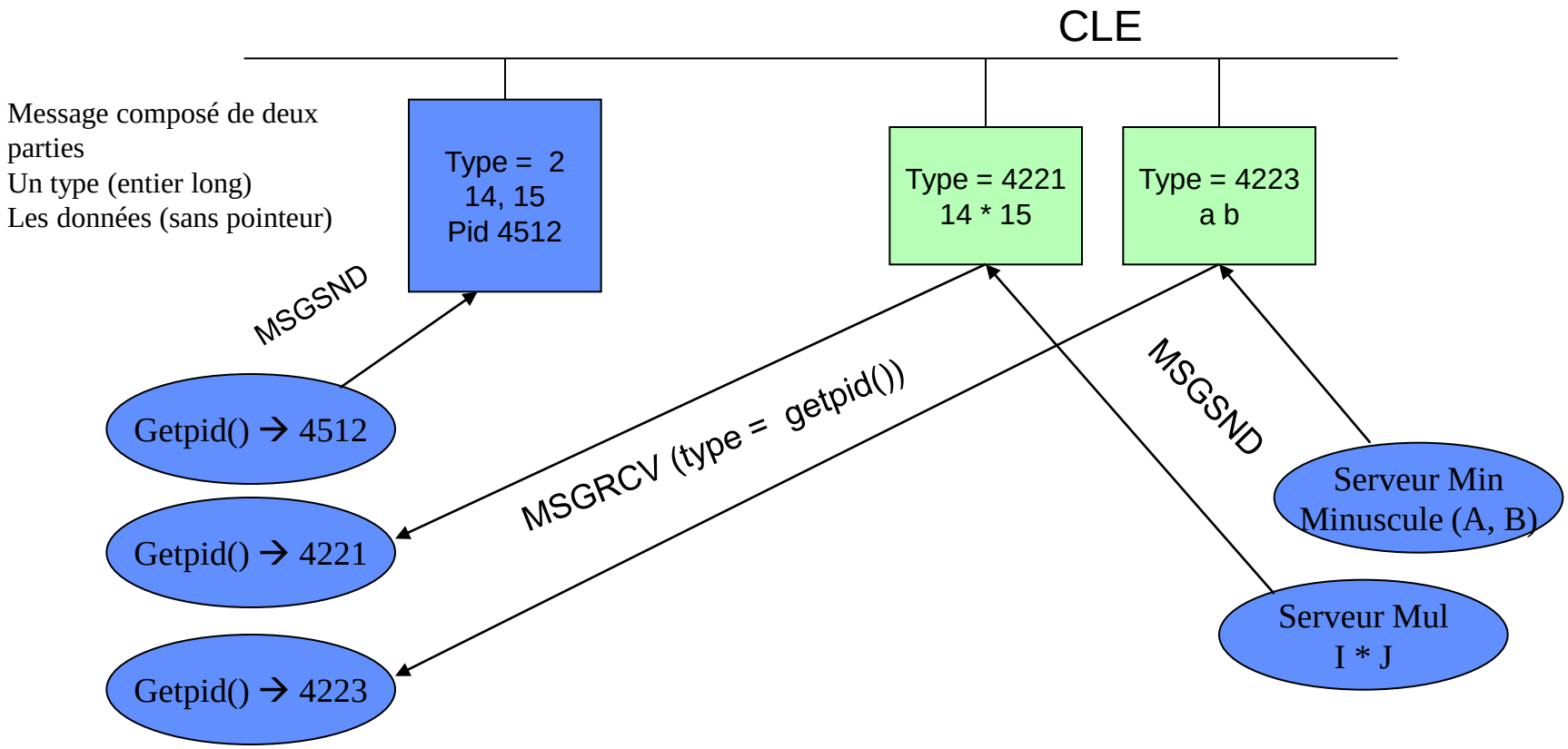
Outils IPC
Les files de message ou MSQ



Le type dans le message permet de désigner le destinataire. La primitive MSGRCV spécifie quel type de message prélever.

Côté client : type à utiliser pour l'identifier de façon unique : son pid
Côté serveur : de petits numéros qui ne peuvent pas correspondre à des pid de clients.

Outils IPC
Les files de message ou MSQ



Le type dans le message permet de désigner le destinataire. La primitive MSGRCV spécifie quel type de message prélever.

Côté client : type à utiliser pour l'identifier de façon unique : son pid
Côté serveur : de petits numéros qui ne peuvent pas correspondre à des pid de clients.

Outils IPC

Listage des IPC : ipcs

ipcs [-s] [-m] [-q] [-u utilisateur]

- Par type de facilité :
  - -q pour afficher les boîtes aux lettres
  - -m pour afficher les segments de mémoire partagée
  - -s pour afficher les ensembles de sémaphore
- Par utilisateur (le créateur) avec l'option -u utilisateur.
- L'exemple suivant illustre le genre de listing fourni par ipcs :

```
bipro: ipcs
 IPC status from /dev/mem as of Mon May 25 10:12:27 DFT 1998
T ID KEY MODE OWNER GROUP
Message Queues:
q 0 0x4107001c -Rrw-rw---- root printq
Shared Memory:
m 0 0x0d050132 --rw-rw-rw- root system
m 126978 0x6605d9da --rw-rw-rw- darmont labo
Semaphores:
s 4096 0x4d09201d --ra-ra---- root system
 s 1 0x620500eb --ra-r--r-- root system
```