

# Processus, Threads et Synchronisation

**M1 - CHPS**

***Architecture Interne des Systèmes d'exploitations (AISE)***

Jean-Baptiste Besnard  
<jean-baptiste.besnard@paratools.com>



Julien Adam  
<julien.adam@paratools.com>

# Programme du Semestre

- ◆ 1 - Généralités sur les OS et Utilisation de base
- ◆ **2 - IO Processus et Chaînage de commandes (et si le temps IPC Sys V)**
- ◆ **3 - Compilation et représentation Binaire**
- ◆ **4 - Architecture Mémoire d'un processus**
- ◆ **5 - Programmation réseau et entrées/sorties avancées**
- ◆ **6 - Virtualisation et Conteneurs**
- ◆ **7 - Noyau Linux et bases d'ordonnancement**
- ◆ **Examen + Démo de projets**

# Les I/Os POSIX

- Les descripteurs de fichier « bas-niveau »
- Les IO de haut niveau FILE\*
- Création de PIPE

**I/Os bas-niveau**

# Les Fichiers Bas-Niveau

## **L'état d'un descripteur de fichier:**

- Droits (lecture ou écriture)
- Bidirectionnel (en fonction des droits)
- Peut correspondre à un flux ou bien un fichier sur le disque
- Possède un offset courant (cas d'un fichier)
- On peut y lire et écrire (Read/Write)

## **Des descripteurs spéciaux:**

- Stdin (0) (ou STDIN\_FILENO de unistd.h) -> Entrée standard
- Stdout (1) (ou STDOUT\_FILENO de unistd.h) -> Sortie standard
- Stderr (2) (ou STDERR\_FILENO de unistd.h) -> Sortie d'erreur standard

## **On peut créer un descripteur avec:**

- Open (sur fichier)
- Pipe (sur flux)
- socket (pour une connection réseau)

# Ouvrir un Fichier

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
int open(const char *path, int oflag, ... /* mode_t mode */);
```

## Soit deux empreintes pour ouvrir:

```
int open(const char *path, int oflag );
```

```
int open(const char *path, int oflag, mode_t mode );
```

## Arguments:

- **path:** chemin vers le fichier
- **oflag:** ou binaire entre les options (O\_RDWR, O\_CREAT, O\_APPEND, ... )
- **(si O\_CREAT) mode:** ou binaire définissant les droits du fichier

**Retour < 0 si erreur**

# Fermer un Fichier

```
#include <unistd.h>
```

```
int close(int fildes);
```

## Arguments:

- **fildes:** descripteur de fichier ouvert

**Retour < 0 si erreur**

# Créer un fichier vide

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int fd = open("./toto", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR );

    if( fd < 0 )
    {
        perror( "open" );
    }

    close(fd);

    return 0;
}
```



# Créer un fichier vide

```
jbbesnard@denéb | <0> | lun. janv. 14 12:17:03  
~/AISE_2  
$ls -la toto  
-rw----- 1 jbbesnard jbbesnard 0 janv. 14 12:16 toto
```

# Créer un fichier vide (non existant)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int fd = open("./toto", O_RDWR | O_CREAT | O_EXCL,
                  S_IRUSR | S_IWUSR );

    if( fd < 0 )
    {
        perror("open");
    }

    close(fd);

    return 0;
}
```

# Créer un fichier vide (non existant)

```
jbbesnard@denéb | <0> | lun. janv. 14 12:19:29  
~/AISE_2  
$ ./a.out  
open: File exists
```

# Lire dans un Fichier

```
#include <unistd.h>
```

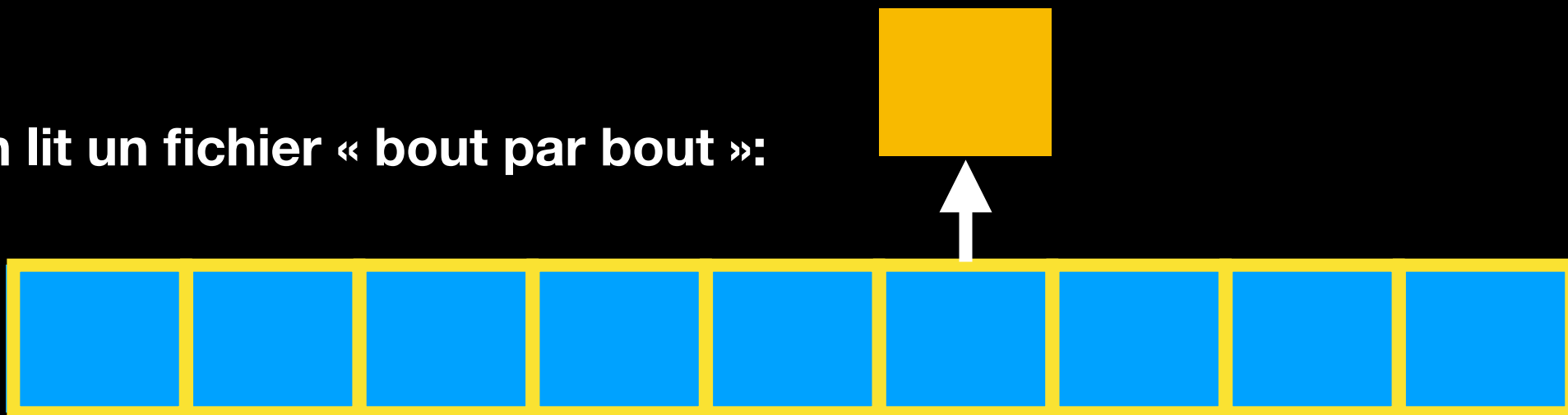
```
ssize_t read(int fd, void *buf, size_t count);
```

**Descripteur**

**Buffer**

**Taille max**

On lit un fichier « bout par bout »:



# Lire dans un Fichier

(possible EAGAIN sur socket)

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>

int main(int argc, char ** argv){

    if( argc != 2 )
    {
        fprintf(stderr, "Usage %s PATH\n", argv[0]);
        return 1;
    }

    int fd = open(argv[1], O_RDONLY);

    if( fd < 0 )
    {
        perror("open");
        return 1;
    }

    ssize_t cnt;
    char buff[500];

    while( (cnt = read(fd, buff, 500)) != 0 )
    {
        if( cnt < 0 )
        {
            perror("read");
            return 1;
        }

    }

    close(fd);

    return 0;
}
```

# Écrire dans un Fichier

```
ssize_t safe_write(int fd, void *buff, size_t size)
{
    size_t written = 0;
    while( (size - written) != 0 )
    {
        errno = 0;
        ssize_t ret = write(fd, buff + written, size-written);

        if( ret < 0 )
        {
            if(errno == EINTR)
            {
                continue;
            }

            perror("write");
            return ret;
        }

        written += ret;
    }
}
```

Symétrie avec Read !

# Écrire dans un Fichier

```
ssize_t safe_write(int fd, void *buff, size_t size)
{
    size_t written = 0;
    while( (size - written) != 0 )
    {
        errno = 0;
        ssize_t ret = write(fd, buff + written, size);

        if( ret < 0 )
        {
            if( (errno == EINTR) )
            {
                continue;
            }

            perror("write");
            return ret;
        }

        written += ret;
    }
}
```

On peut aussi vouloir gérer un signal entrant EINTR

# Liste des Cas

## Les cas à gérer pour read bas niveau:

- **EOF** : la fin du fichier (retour 0)
- **>0** : N bytes on été lus (**peut être moins que SIZE!!**)
- **<0** : Une erreur s'est produite
  - ➔ `errno == EINTR` l'appel a été interrompu par un signal
  - ➔ `errno == EAGAIN` si on a marqué le fd `O_NONBLOCK`

## Les cas à gérer pour write bas niveau:

- **>0** : N bytes on été écrits (**peut être moins que SIZE!!**)
- **<0** : Une erreur s'est produite
  - ➔ `errno == EINTR` l'appel a été interrompu par un signal
  - ➔ `errno == EAGAIN` si on a marqué le fd `O_NONBLOCK`



# Changer d'Offset

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
off_t lseek(int fd, off_t offset, int whence);
```

Whence	Description
SEEK_SET	Règle l'offset à « offset »
SEEK_CUR	Retourne l'offset courant plus le paramètre offset
SEEK_END	Retourne l'offset de fin plus le paramètre offset

# Se déplacer à la Fin d'un Fichier



`lseek(fd, 0, SEEK_END);`



Un équivalent ?

# Un équivalent ?

Ouvrir le FD avec le paramètre  
O\_APPEND dans le Open !



R.T.⚠.M.

**open(2), close(2), stat(2), read(2), write(2),  
pread(2), pwrite(2), fsync(2)**

**Avancé : fcntl(2)**

**UMASK**

# Notion de UMASK

**Il est possible régler un masque de droit par défaut:**

- Pour automatiquement masquer certain droits sur les nouveaux fichiers
- Ces droits s'appliquent à tout fichier nouvellement créé et s'écrivent en octal

```
jbbesnard@denéb | <0> | lun. janv. 14 12:28:38  
~/AISE_2  
$umask  
0022
```

**Quels droits sont masqués ?**

Valeur	R	W	X
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

# Notion de UMASK

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int fd = open("./toto_um", O_RDWR | O_CREAT, 0777 );

    if( fd < 0 )
    {
        perror( "open" );
    }

    close(fd);

    return 0;
}
```

Umask 0022 quels sont les droits de toto\_um ?



# Notion de UMASK

```
jbbesnard@denéb | <0> | lun. janv. 14 12:35:22  
~/AISE_2  
$ls -lah toto_um  
-rwxr-xr-x 1 jbbesnard jbbesnard 0 janv. 14 12:34 toto_um
```

# Notion de UMASK

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    umask(0777);
    int fd = open("./toto_um", O_RDWR | O_CREAT, 0777 );

    if( fd < 0)
    {
        perror("open");
    }

    close(fd);

    return 0;
}
```

Quels sont les droits de toto\_um ?

# Notion de UMASK

```
jbbesnard@deneb | <0> | lun. janv. 14 12:38:00  
~/AISE_2  
$ls -lah toto_um  
----- 1 jbbesnard jbbesnard 0 janv. 14 12:37 toto_um
```

# Notion de UMASK

Comment régler les droits totalement dans OPEN ?

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int oldm = umask(0000);
    int fd = open("./toto_um", O_RDWR | O_CREAT, 07777 );
    umask(oldm);

    if( fd < 0 )
    {
        perror("open");
    }

    close(fd);

    return 0;
}
```

# Calcul du UMASK

$$D = \text{PERM} \& (\sim \text{UMASK})$$

Exemple:

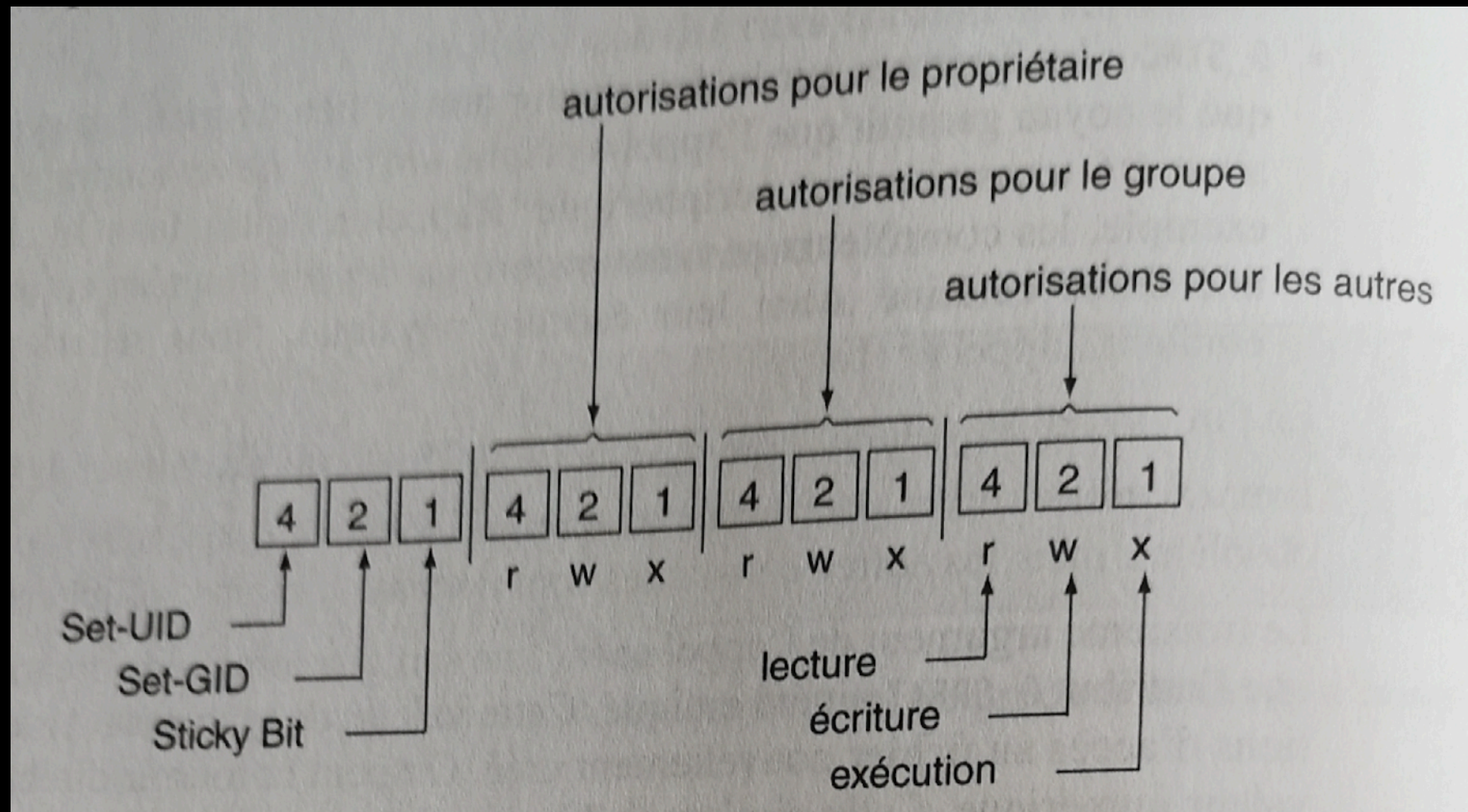
$$D = 0777 \& (\sim 0022)$$

$$D = 111\ 111\ 111 \& (\sim 000\ 010\ 010)$$

$$D = 111\ 111\ 111 \& 111\ 101\ 101$$

$$D = 111\ 101\ 101$$

# Droits en Détail



**set-UID:** execution possible avec l'utilisateur du binaire

**set-GID:** execution possible avec le groupe du binaire

**Sticky bit (sur repertoire):** seul le propriétaire du répertoire et du fichier peuvent le supprimer utilisé pour /tmp

**I/Os Haut Niveau**

# Ouvrir un Fichier (FILE \*)

```
#include <stdio.h>
```

```
FILE *fopen(const char *pathname, const char *mode);
```

```
FILE *fdopen(int fd, const char *mode);
```

**Le FILE \* est une sur-couche au FD précédemment vu.**



# Fermer un Fichier (FILE \*)

```
#include <stdio.h>
```

```
int fclose(FILE *stream);
```

# Ouvrir un Fichier (FILE \*)

```
#include <stdio.h>

int main(int argc, char ** argv){

    FILE * fd = fopen(argv[1], "r");

    if(!fd){
        perror("fopen");
        return 1;
    }

    fclose(fd);

    return 0;
}
```

# Lire un Fichier

```
#include <stdio.h>

int main(int argc, char ** argv){
    FILE * fd = fopen(argv[1], "r");
    if(!fd){
        perror("fopen");
        return 1;
    }
    char buff[500];
    size_t cnt;

    while( 1 )
    {
        cnt = fread(buff, sizeof(char), 500, fd);
        if( cnt == 0)
        {
            if( feof(fd) )
            {
                break;
            }
            else
            {
                perror("fread");
                return 1;
            }
        }

        /* USE your buff here */
    }
    fclose(fd);
    return 0;
}
```

# Lire ligne par ligne

```
#include <stdio.h>

int main(int argc, char ** argv){

    if( argc != 2 )
        return 1;

    FILE * fd = fopen(argv[1], "r");

    if(!fd){
        perror("fopen");
        return 1;
    }

    char buff[500];
    char * ret;

    while(1)
    {
        ret = fgets(buff, 500, fd);

        if(!ret)
        {
            if( feof(fd) )
            {
                /* EOF all OK*/
                break;
            }
            else
            {
                /* Error */
                perror("fgets");
                return 1;
            }
        }

        /* USE your buff here */
        fprintf(stdout, "%s", ret );
    }
    fclose(fd);
    return 0;
}
```

# Ecrire dans un Fichier

```
#include <stdio.h>
#include <string.h>

int main(int argc, char ** argv){
    if(argc != 2 )
        return 1;

    FILE * fd = fopen(argv[1], "w");

    if(!fd){
        perror("fopen");
        return 1;
    }

    char data[] = "Hello I/Os\n";
    size_t cnt;

    cnt = fwrite(data, sizeof(char),
                  strlen(data), fd);

    if( cnt == 0)
    {
        perror("fread");
        return 1;
    }

    fclose(fd);

    return 0;
}
```



**RTM**  
Before you ask those kinds of questions.

**fopen, fclose, fread, fwrite, fgetc, fputc, fseek, feof, fileno, fdopen**

# Redirection de Flux

# Redirection de Flux

```
#include <unistd.h>
```

```
int dup2(int oldfd, int newfd);
```

**Dup2 remplace « newfd » par « oldfd » et se charge de fermer « newfd ».**



# Exemple

## Redirection de sortie dans un fichier

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char ** argv )
{
    pid_t child = fork();

    if( child == 0)
    {
        int out = open("./out.dat", O_CREAT | O_WRONLY ,
                        0600);
        /* Replace stdout with the file */
        dup2(out, STDOUT_FILENO);
        close(out);
        char * argv[] = {"ls", "-la", NULL};
        execvp( argv[0], argv);
    }
    else
    {
        /* Parent closes out */
        wait(NULL);
    }

    return 0;
}
```

# Création de Pipe

```
#include <unistd.h>
```

```
int pipe(int pipefd[2]);
```

**Crée un « tuyau » == PIPE en anglais.**

```
pipefd[2] = { READ_END, WRITE_END };
```



**Un pipe est UNIDIRECTIONNEL**

# Chainer deux Commandes

```
echo "Salut Tout Le Monde " | tac -s " "
```

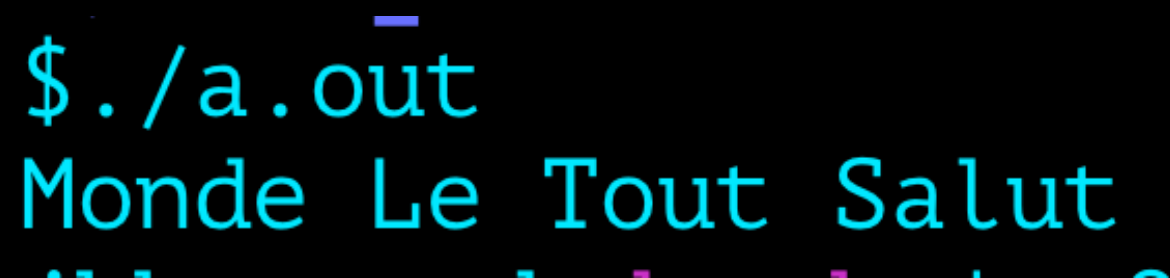
```
$echo "Salut Tout Le Monde " | tac -s " "
```

```
Monde Le Tout Salut
```

```
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
```

# Chainer deux Commandes

echo "Salut Tout Le Monde " | tac -s " "

  
\$ ./a.out  
Monde Le Tout Salut

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char ** argv )
{
    int pp[2];
    pipe(pp);

    pid_t child1 = fork();

    if( child1 == 0)
    {
        /* Replace stdout with the write end of the pipe */
        dup2(pp[1], STDOUT_FILENO);
        /* Close read end of the pipe */
        close(pp[0]);
        /* Run command */
        char * argv[] = {« printf", "Salut Tout Le Monde « , NULL};
        execvp( argv[0], argv);
    }
    else
    {
        pid_t child2 = fork();

        if(child2 == 0)
        {
            /* Replace stdin with the read end of the pipe */
            dup2(pp[0], STDIN_FILENO);
            /* Close write end of the pipe */
            close(pp[1]);
            /* Run command */
            char * argv[] = {"tac", "-s", " ", NULL};
            execvp( argv[0], argv);
        }
        else
        {
            /* Close both end of the pipe */
            close(pp[0]);
            close(pp[1]);
            /* wait for two child */
            wait(NULL);
            wait(NULL);
        }
    }

    return 0;
}
```

# Chainer deux Commandes

In

./a.out

Out  
pp[2]

In

echo

Out  
pp[2]

In

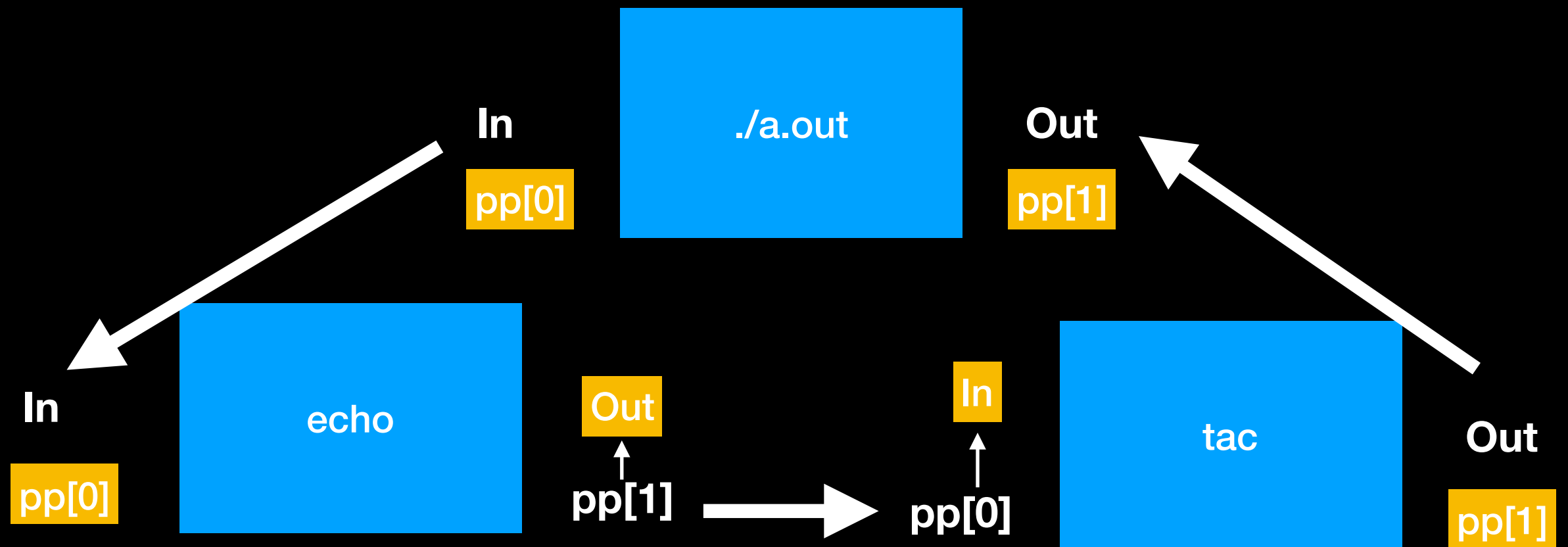
tac

Out  
pp[2]

**Juste après le fork,  
les descripteurs sont dans  
tous les fichiers.**

# Chainer deux Commandes

Ensuite on insère le PIPE entre les deux commandes.



# Généralités sur les IPC System V

# Les IPC System V

Apparus dans Unix en 1983 ils permettent des communication inter-inter-processus (Inter-Process Communications, IPC)

- Files de messages
- Segment de mémoire partagée
- Sémaphores

Le noyau est chargé de la gestion des ressources associées via des commandes

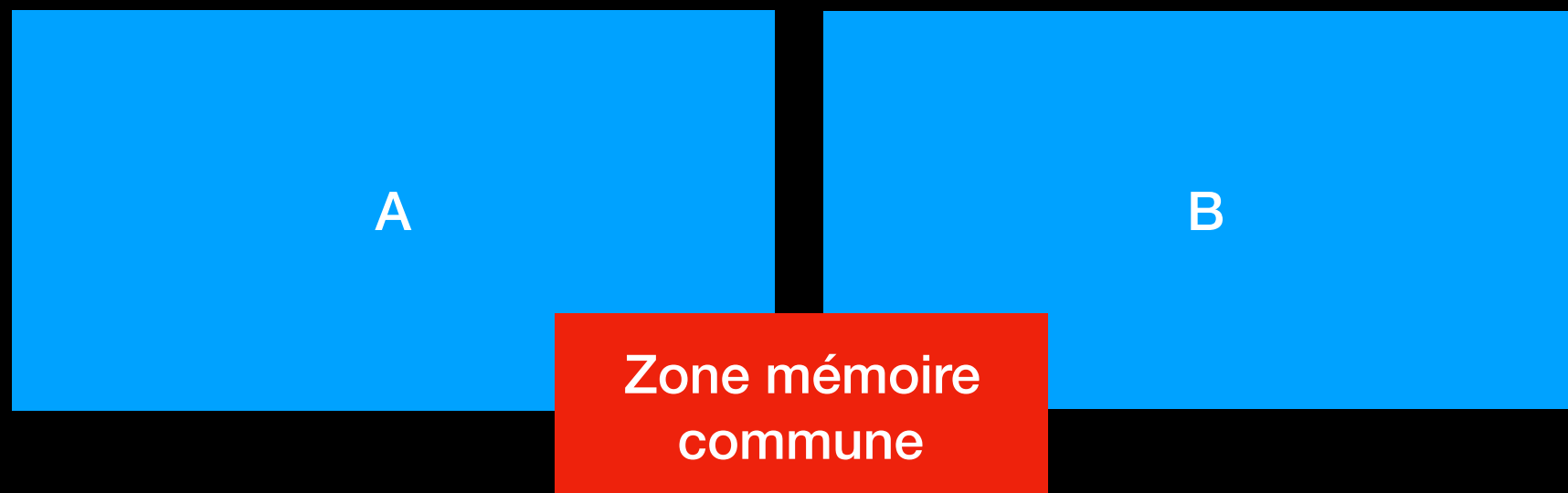


# Files de Messages



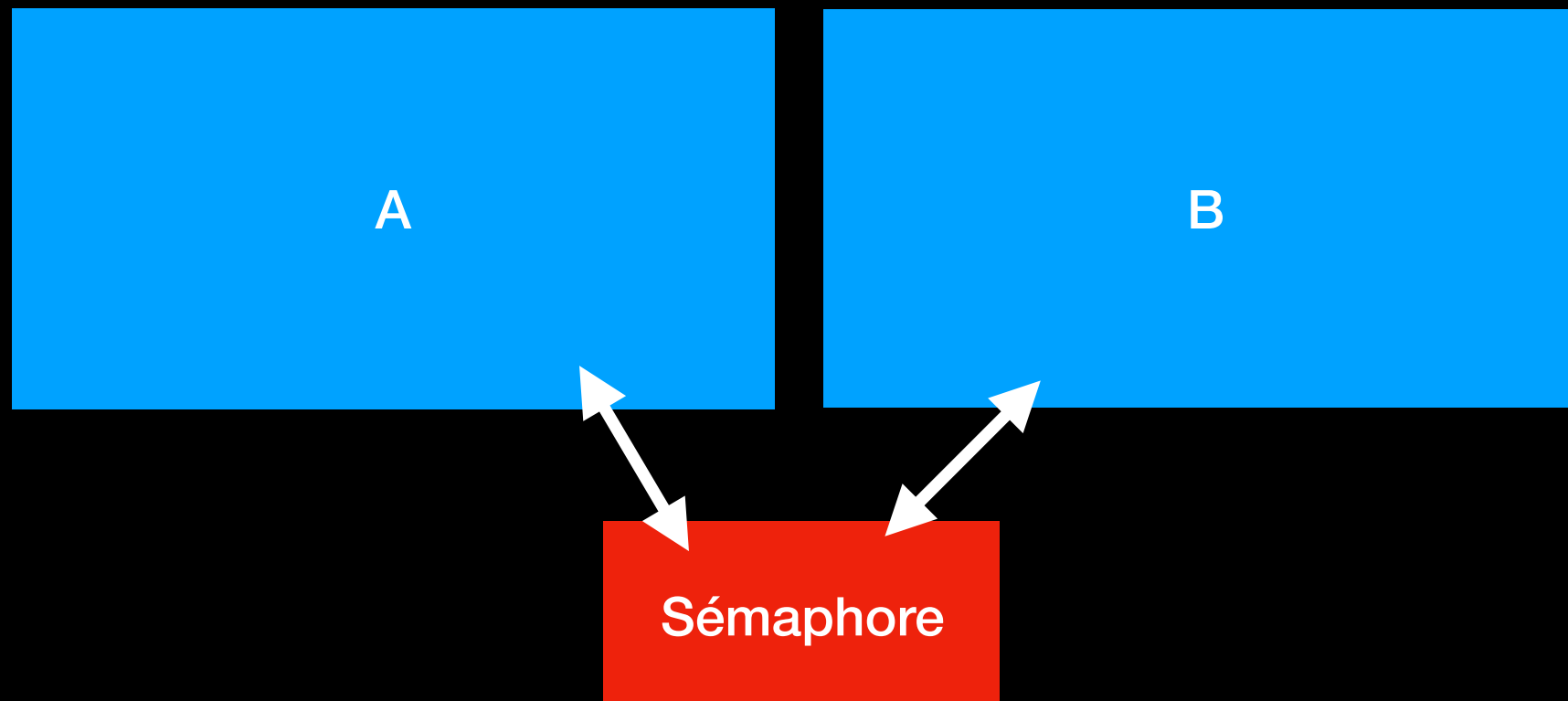
- *ftok*: génération d'une clef IPC
- *msgget*: Récupère un identificateur de file de message
- *msgrecv*: Réception d'un message depuis une file
- *msgsend*: Envoi d'un message dans une file
- *msgctl*: Contrôle de la file de messages

# Segment de mémoire partagée



- *ftok*: génération d'une clef IPC
- *shmget*: Récupère un identificateur de segment shm
- *shmat*: Projection d'un segment SHM
- *shmdt*: Supression d'un segment she
- *shmctl*: Contrôle du segment SHM

# Sémaphore IPC



- *ftok*: génération d'une clef IPC
- *semget*: Récupère un identificateur de sémaphore
- *semop*: Fait une opération sur le sémaphore
- *semctl*: Contrôle du sémaphore

# Les IPC System V

**\$ ipcs**

----- Files de messages -----

clef	msqid	propriétaire	perms	octets utilisés	messages
------	-------	--------------	-------	-----------------	----------

----- Segment de mémoire partagée -----

clef	shmid	propriétaire	perms	octets	nattch	états
0x00000000	42729472	jbbsnard	600	1048576	2	dest
0x00000000	39616513	jbbsnard	600	524288	2	dest

----- Tableaux de sémaphores -----

clef	semid	propriétaire	perms	nsems
------	-------	--------------	-------	-------

# Les IPC System V

```
$ ipcrm -h
```

Utilisation :

```
ipcrm [options]  
ipcrm shm|msg|sem <id> ...
```

Supprimer certaines ressources IPC.

Options :

```
-m, --shm-id <ident.>    retirer le segment de mémoire partagée par ident.  
-M, --shm-key <clef>     retirer le segment de mémoire partagée par clef  
-q, --queue-id <ident.>  retirer la file de messages par identifiant  
-Q, --queue-key <clef>   retirer la file de messages par clef  
-s, --semaphore-id <id.> retirer le sémaphore par identifiant  
-S, --semaphore-key <clef> retirer le sémaphore par clef  
-a, --all[=shm|msg|sem]  tout retirer (dans la catégorie indiquée)  
-v, --verbose            expliquer les actions en cours  
  
-h, --help              afficher cette aide et quitter  
-V, --version            afficher les informations de version et quitter
```

Consultez ipcrm(1) pour obtenir des précisions complémentaires.

# Les IPC System V

```
$ ipcmk -h
```

Utilisation :  
ipcmk [options]

Créer diverses ressources IPC.

Options :

-M, --shmem <taille>	créer un segment de mémoire partagée de taille <taille>
-S, --semaphore <nsems>	créer un tableau de sémaphores à <nsems> éléments
-Q, --queue	créer une file de messages
-p, --mode <mode>	droits de la ressource (0644 par défaut)
-h, --help	afficher cette aide et quitter
-V, --version	afficher les informations de version et quitter

Consultez ipcmk(1) pour obtenir des précisions complémentaires.

# Resources

**Les resources IPC sont indépendante des processus**

- Il est possible de laisser des scories si l'on ne fait pas attention
- Un processus peut se « rater » à un segment lors de son redémarrage par exemple
- Les processus partagent des segments avec un mécanisme de clef qui est un secret « a priori » pour la sécurité

# Clefs pour les IPCs System V



# La Clef

**Un IPC (de tout type) est partagé par une clef:**

- C'est un entier qui doit être le même entre tous les processus partageant la resource;
- On peut la connaître a priori avec risque de conflit (un peut comme un port TCP);
- Une clef spéciale IPC\_PRIVATE crée une file limité à un processus et l'ensemble de ses descendants;
- On peut la créer avec une fonction « ftok » qui repose sur un fichier et un nom de projet.

# Ftok

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
```

```
key_t ftok(const char *pathname, int proj_id);
```

## DESCRIPTION

The `ftok()` function uses the identity of the file named by the given `pathname` (which must refer to an existing, accessible file) and the least significant 8 bits of `proj_id` (which must be nonzero) to generate a `key_t` type System V IPC key, suitable for use with `msgget(2)`, `semget(2)`, or `shmget(2)`.

The resulting value is the same for all `pathnames` that name the same file, when the same value of `proj_id` is used. The value returned should be different when the (simultaneously existing) files or the project IDs differ.

## RETURN VALUE

On success, the generated `key_t` value is returned. On failure `-1` is returned, with `errno` indicating the error as for the `stat(2)` system call.



# Création / Récupération de ressources

Une fois que l'on a une clef de type *key\_t* on peut retrouver/créer une resource:

- File de message : *msgget*
- Segment de mémoire partagée: *shmget*
- Sémaphore: *semget*

# Les Files de Messages

## IPC SYSTEM V

# Files de Messages pour une Communication entre Processus sur un Même Noeud.

Le message sera toujours de la forme:

```
Struct XXX {  
    long id; // Toujours > 0 !  
    ... DATA ...  
    // Taille max sans le long MSGMAX (8192 Octets)  
};
```

Lors de l'envoi et de la réception d'un message la taille et TOUJOURS sans le long qui définit le type de message. Cette même valeur (ici id) doit TOUJOURS être supérieure à 0.

En pratique on crée une struct statique sur la pile car l'allocation d'un objet avec piggybacking demande plus de code.

# Créer Une File de Messages

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgget(key_t key, int msgflg);
```

- Key : Une clef, soit manuelle, soit via ftok ou bien IPC\_PRIVATE
- msgflg: mode de création de la file et ses droits UNIX
  - ➡ IPC\_CREAT crée une file s'il y en a aucune associée à cette clef
  - ➡ IPC\_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC\_CREAT!)
  - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file et moins pratique !)

# Créer Une File de Messages

- Créer une file pour un processus et ses fils

➡ `file = msgget(IPC_PRIVATE, 0600);`

- Créer une file pour accéder à une file potentiellement existante:

➡ `file = msgget(key , IPC_CREAT | 0600);`

- Pour être sûr de créer une nouvelle file en lecture écriture pour soi et en lecture seule pour les autres utilisateurs:

➡ `file = msgget(key, IPC_CREAT | IPC_EXCL | 0622);`

- Utiliser uniquement une file existante précédemment créée par un serveur:

➡ `file = msgget(key, 0);`

# Envoyer un Message

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

- msqid : file de message à utiliser, créée avec msgget
- msgp : pointeur vers les données à envoyer (comprend forcément un long qui est l'ID de message)
- size : taille du message **SANS** le long qui est l'ID du message
- msgflg: mode d'envoi du message
  - ➡ IPC\_NOWAIT ne pas bloquer si la file est pleine (renvoie EAGAIN dans errno)
  - ➡ 0 en général



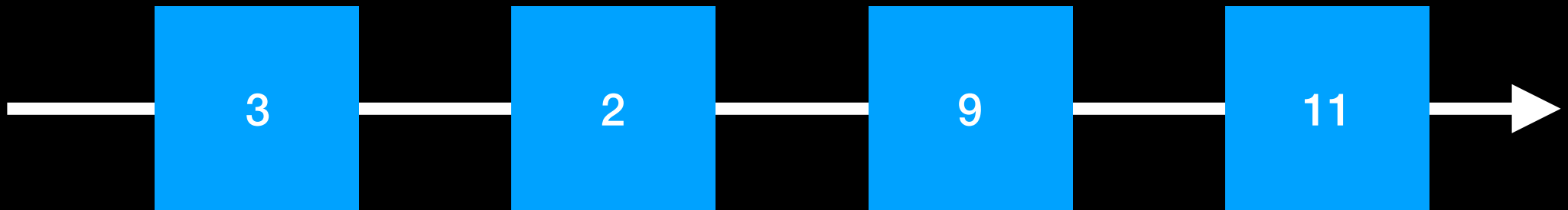
# Recevoir un Message

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
ssize_t msgrcv(int msqid,
               void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

- msqid : file de message à utiliser, créée avec msgget
- msgp : pointeur vers les données à envoyer (comprend forcément un long qui est l'ID de message)
- size : taille du message **SANS** le long qui est l'ID du message
- msgtyp : type de message à recevoir:
  - ➡ 0 : prochain message de la file
  - ➡ 0 < TYP prochain message avec l'ID donné
  - ➡ TYP < 0 prochain message avec un ID inférieur ou égal à TYP, utilisé pour gérer des priorités de messages
- msgflg: mode de réception du message:
  - ➡ IPC\_NOWAIT ne pas bloquer si pas de message du TYP donné (renvoie ENOMSG dans errno)
  - ➡ MSG\_EXCEPT renvoie un message d'un TYP différent de celui donné (seulement pour TYP > 0)
  - ➡ MSG\_NO\_ERROR permettre au message d'être tronqué à la réception (à la différence du comportement de base)

# Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 2, 0);
```

**Quel message ??**

# Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 2, 0);
```

## Quel message ??

2

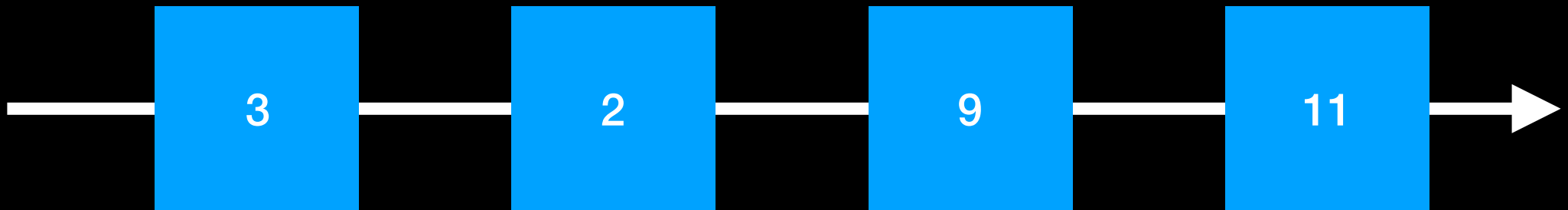
# Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), -10, 0);
```

## Quel message ??

# Recevoir un Message

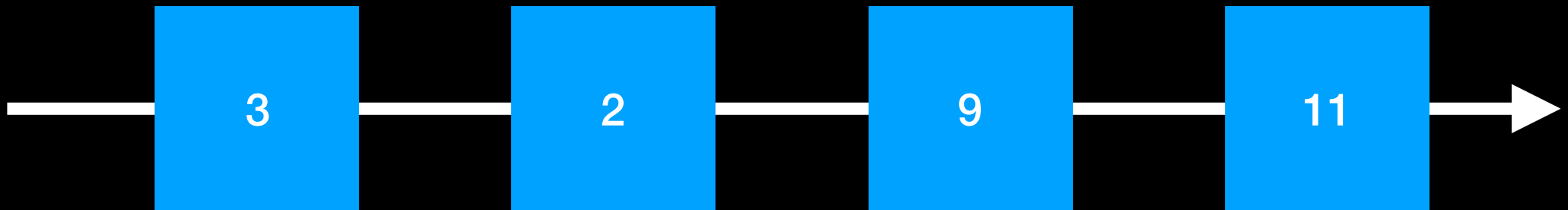


```
msgrcv(file, &msg, sizeof(int), -10, 0);
```

## Quel message ??

9

# Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 99, 0);
```

**Quel message ??**

# Recevoir un Message

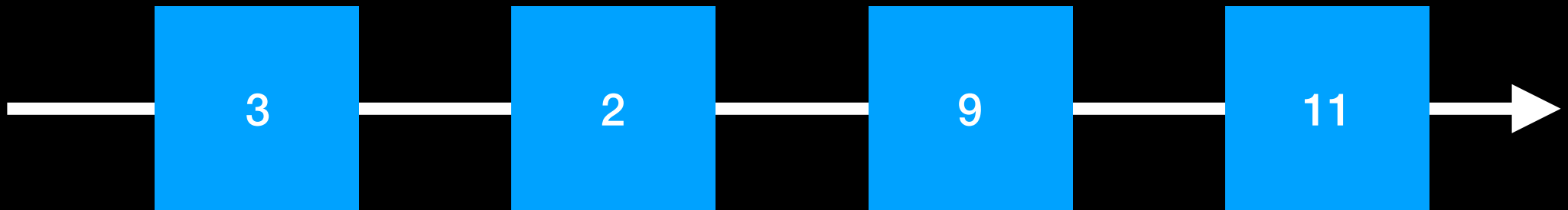


```
msgrcv(file, &msg, sizeof(int), 99, 0);
```

## Quel message ??

L'appel reste bloqué indéfiniment si un message 99 n'est jamais posté.

# Recevoir un Message

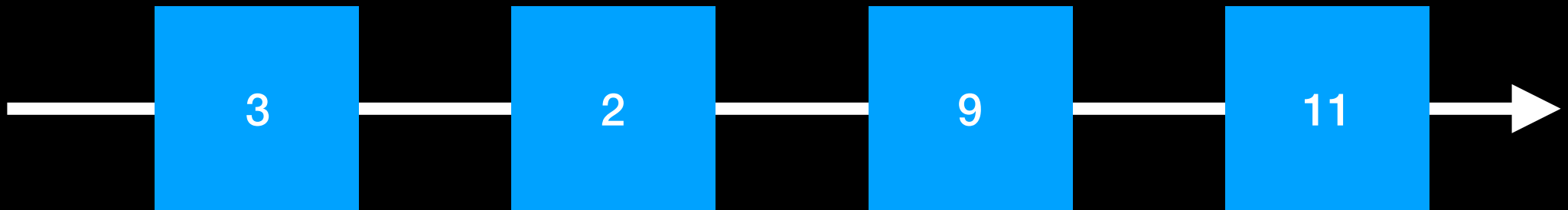


```
msgrcv(file, &msg, sizeof(int), 99, IPC_NOWAIT);
```

## Quel message ??



# Recevoir un Message

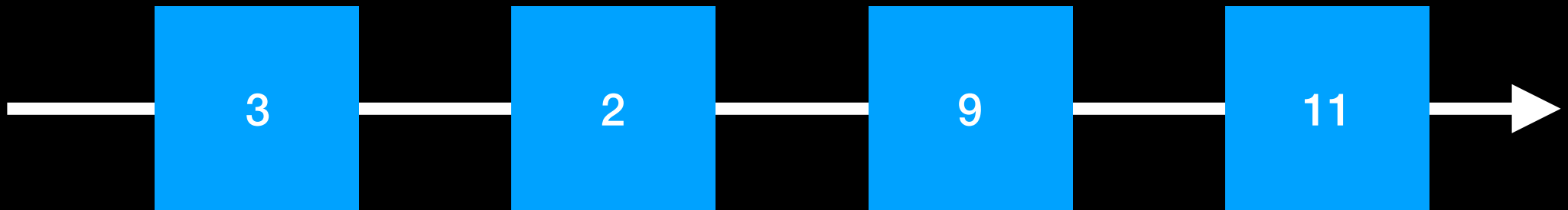


```
msgrcv(file, &msg, sizeof(int), 99, IPC_NOWAIT);
```

## Quel message ??

L'appel renvoie -1 et met errno à ENOMSG

# Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 11, MSG_EXCEPT);
```

## Quel message ??

# Recevoir un Message



```
msgrcv(file, &msg, sizeof(int), 11, MSG_EXCEPT);
```

## Quel message ??

9

# Contrôler une File

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- msqid : ID de la file à contrôler
- cmd: commande à appliquer à la file
  - ➡ IPC\_STAT récupères les informations sur la file dans la *struct msqid\_ds* (voir man)
  - ➡ IPC\_SET permet de régler certains attributs en passant une *struct msqid\_ds*
  - ➡ **IPC\_RMID supprime la file toute les opérations courantes ou future échouent (avec la possibilité non gérée qu'une nouvelle file soit créée avec la même clef). La synchronisation et à la charge du programmeur.**
  - ➡ ... il existe d'autre flags voir man

## PENSEZ à SUPPRIMER VOS FILES !!!

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <sys/wait.h>

#include <sys/time.h>

double get_time(){
    struct timeval val;
    gettimeofday(&val, NULL);
    return (double)val.tv_sec + 1e-6 * val.tv_usec;
}

#define SIZE 16

struct msg_t{
    long type;
    int data[SIZE];
};

#define NUM_MSG 65536

int main( int argc, char ** argv ){
    int file = msgget(IPC_PRIVATE, IPC_CREAT | 0600);

    if( file < 0 ){
        perror("msgget");
        return 1;
    }

    int i;
    struct msg_t m;
    m.type = 1;

    int pid = fork();

    if( pid == 0 )
    {
        int stop = 0;

        while(!stop)
        {
            msgrcv(file, &m, SIZE*sizeof(int), 0, 0);
            /* Notify end */
            if( m.data[0] == 0 )
                stop = 1;
            m.type = 1;
            msgsnd(file, &m, SIZE*sizeof(int), 0);
        }
    }
}

```

```

else
{
    double total_time = 0.0;

    for( i = 2 ; i <= NUM_MSG ; i++)
    {
        m.data[0] = i;
        m.type = i;

        double start = get_time();
        int ret = msgsnd(file, &m, SIZE*sizeof(int), 0);

        if( ret < 0 )
        {
            perror("msgsend");
            return 1;
        }

        double end = get_time();
        total_time += end - start;

        msgrcv(file, &m, SIZE*sizeof(int), 1, 0);
    }

    m.data[0] = 0;
    msgsnd(file, &m, SIZE*sizeof(int), 0);

    wait( NULL );

    msgctl( file, IPC_RMID, NULL);

    fprintf(stderr, "Pingpong takes %g usec Bandwidth is %g MB/s\n",
        total_time/NUM_MSG*1e6,
        (double)(SIZE*NUM_MSG*sizeof(int))/
            (total_time*1024.0*1024.0));

}

return 0;
}

```

# Les Segments SHM

## IPC SYSTEM V

# Partager une Zone Mémoire entre Deux Processus

## SHM = SHared Memory

### Les avantages:

- Communication directe sans recopie mémoire;
- Pas de passage par l'espace noyau à la différence des files messages (context switch et recopie);
- Latences plus faible (même mémoire)

### Les inconvénients:

- Il faut manuellement synchroniser les communications (lock ou sémaphore)
  - ➡ *Comprenez qu'il est possible de mettre un lock dans cette zone mémoire, un spin lock directement, un mutex avec le bon attribut (PTHREAD\_PROCESS\_SHARED). Ou bien un sémaphore des IPC.*
- La structuration des données est à la charge du programme

# Créer le Segment SHM

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmget(key_t key, size_t size, int shmflg);
```

- key : Une clef, soit manuelle, soit via ftok ou bien IPC\_PRIVATE
- Size: taille du segment SHM en octet (arrondie à la page supérieure).  
**Donc mapper un int est un gros gâchis de mémoire (une page fait 4 KB).**
- shmflg: mode de création de la file et ses droits UNIX
  - ➡ IPC\_CREAT crée une file s'il y en a aucune associée à cette clef
  - ➡ IPC\_EXCL échoue s'il existe déjà une file sur la clef indiquée (toujours combiné avec IPC\_CREAT!)
  - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file est moins pratique !)



# Projeter le Segment SHM

```
#include <sys/types.h>
#include <sys/shm.h>
```

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- shmid : le descripteur du segment SHM
- shmaddr: une adresse où mapper le segment, alignée sur une frontière de page. **NULL si indifférent.**
- shmflg: options relative à la projection du segment
  - ➡ SHM\_RND arrondis l'adresse passée par *shmaddr* à une frontière de page
  - ➡ SHM\_RDONLY partager le segment en lecture seule
  - ➡ ... il existe d'autres flags voir man

# Retirer le Segment SHM

```
#include <sys/types.h>
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

- shmaddr: adresse renvoyée par shmat

**Tous les processus doivent retirer le segment de leur mémoire autrement la suppression avec shmctl n'est pas effective. Si un processus se termine il détache la mémoire mais cela ne marque pas le segment pour suppression.**

# Supprimer le Segment SHM

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- shmid : ID du segment à contrôler
- cmd: commande à appliquer à la file
  - ➡ IPC\_STAT récupères les informations sur la file dans la *struct shmid\_ds* (voir man)
  - ➡ IPC\_SET permet de régler certains attributs en passant une *struct shmid\_ds*
  - ➡ **IPC\_RMID marque le segment SHM pour destruction cela ne se produira que quand tout les processus l'ayant projeté se seront détachés**
  - ➡ ... il existe d'autre flags voir man particulièrement IPC\_INFO et SHM\_INFO utiles pour connaitre les limites sur le système cible

## PENSEZ à SUPPRIMER VOS Segments !!!

## Totalement arbitraire

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
#include <unistd.h>
#include <stdio.h>
```

```
int main(int argc, char **argv)
{
    int shm = shmget(19999, 2 * sizeof(int),
                    IPC_CREAT | IPC_EXCL | 0600 );

    if( shm < 0 )
    {
        perror("shmget");
        return 1;
    }

    int *val = (int*) shmat(shm, NULL, 0);

    if( !val )
    {
        perror("shmat");
        return 1;
    }

    /* valeur de départ */
    val[0] = 1;
    val[1] = 0;

    while(val[0])
    {
        sleep(1);
        val[1]++;
    }

    /* Unmap segment */
    shmdt(val);
    /* Server marks the segment for deletion */
    shmctl(shm, IPC_RMID, NULL);

    return 0;
}
```

# Serveur

```
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
#include <unistd.h>
#include <stdio.h>
```

```
int main(int argc, char **argv)
{
    int shm = shmget(19999, 2 * sizeof(int), 0 );

    if( shm < 0 )
    {
        perror("shmget");
        return 1;
    }

    int *val = (int*) shmat(shm, NULL, 0);

    if( !val )
    {
        perror("shmat");
        return 1;
    }

    /* valeur de départ */
    int last_val = -1;
    while(1)
    {
        if( val[1] != last_val ){
            printf("Val is %d max is 60\n", val[1]);
            last_val = val[1];

            /* Stop condition */
            if( 60 <= val[1] )
            {
                val[0] = 0;
                break;
            }
        }
        else
        {
            usleep(100);
        }
    }

    /* Unmap segment */
    shmdt(val);

    return 0;
}
```

# Client

```
$ ./serveur &  
$ ipcs -m
```

```
----- Segment de mémoire partagée -----  
clef      shmid      propriétaire perms      octets      nattch      états  
0x00004e1f 42827778      jbbesnard  600        8           1
```

```
$ ./client
```

```
Val is 0 max is 60  
Val is 1 max is 60  
(...)  
Val is 7 max is 60  
Val is 8 max is 60  
Val is 60 max is 60
```

```
[2]+  Fini                  ./server
```

```
$ ipcs -m
```

```
----- Segment de mémoire partagée -----  
clef      shmid      propriétaire perms      octets      nattch      états
```

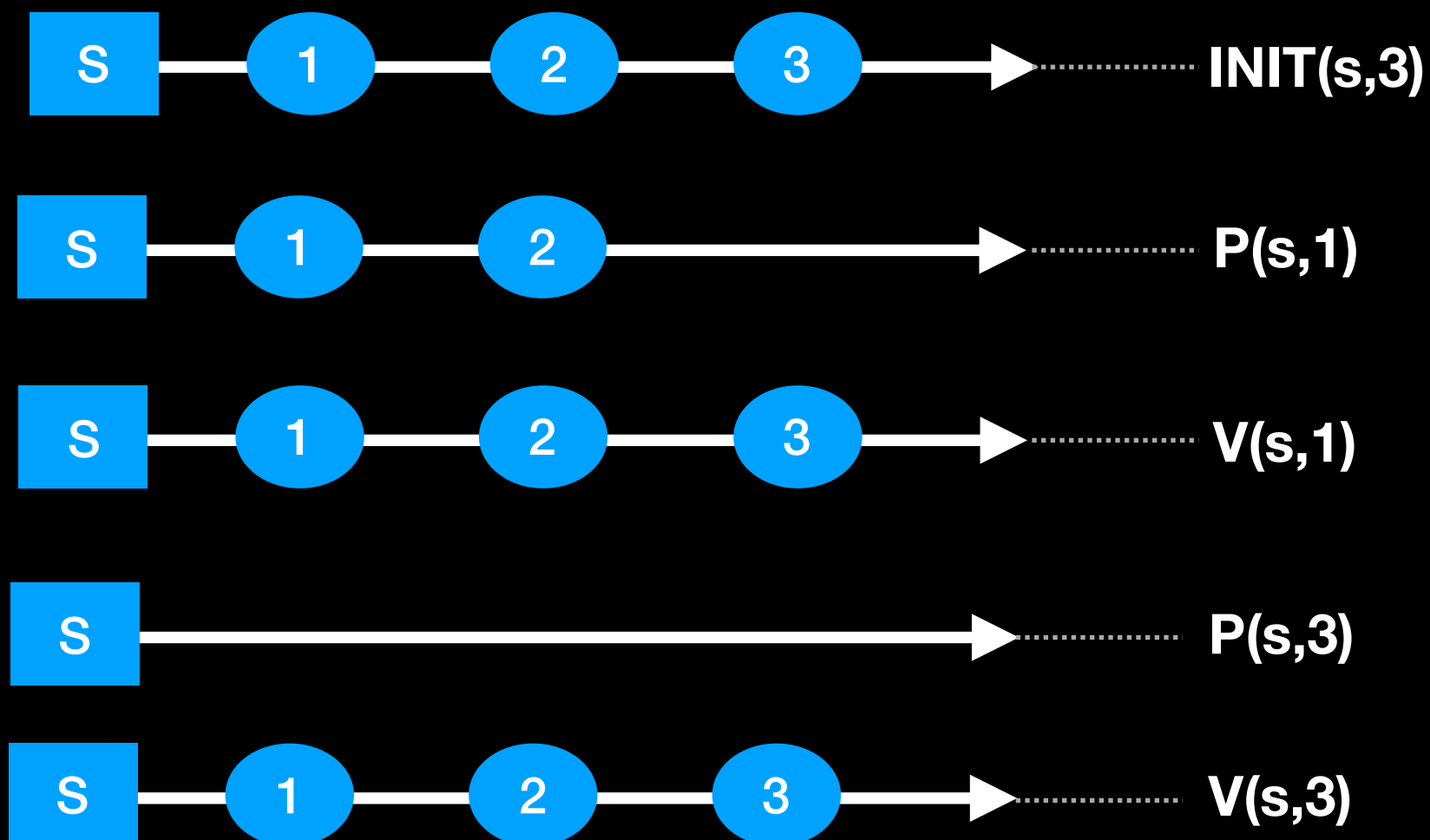
# Les Sémaphores

## IPC SYSTEM V

# Notion de Sémaphore

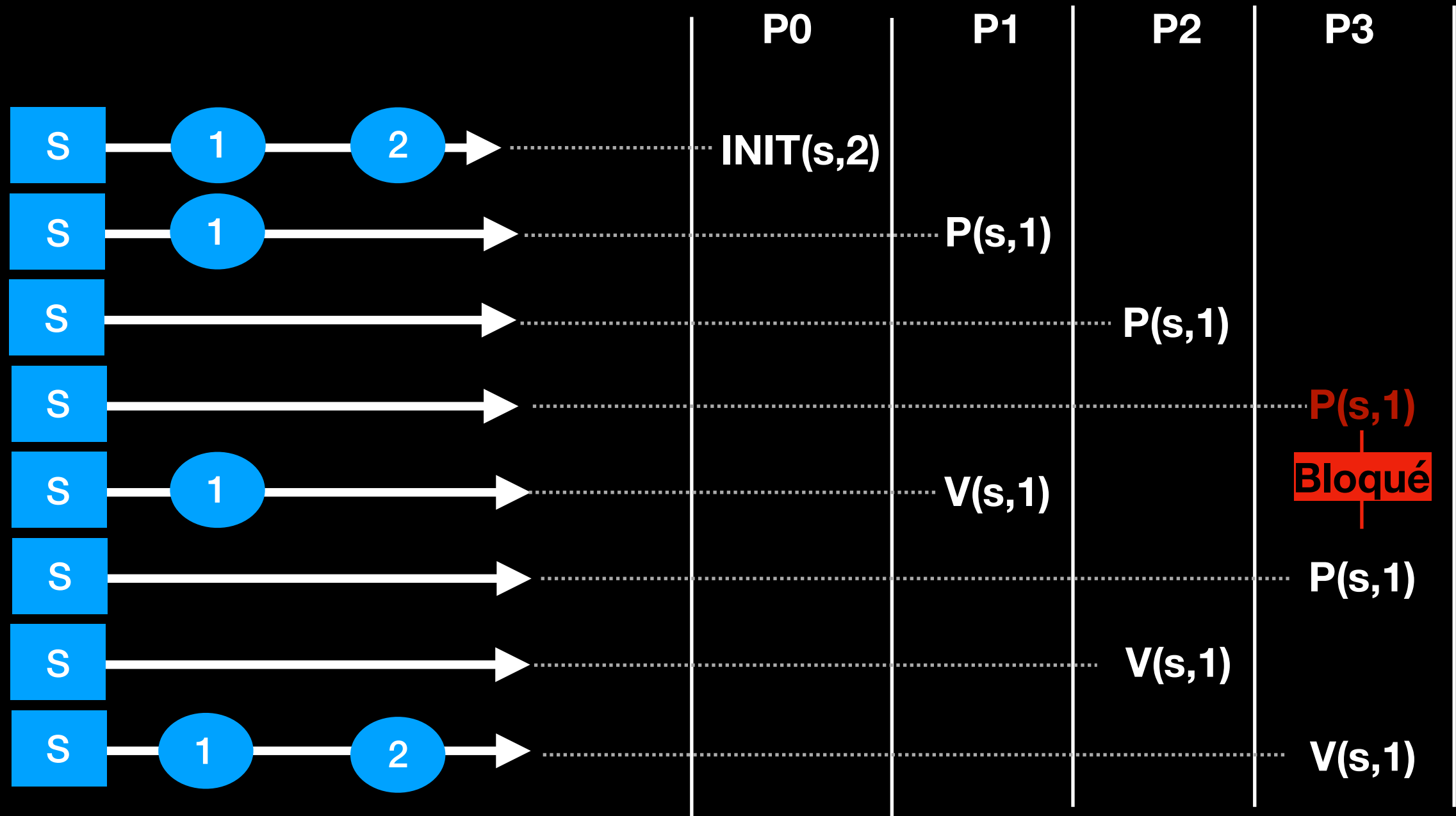
Un sémaphore est un élément de synchronisation qui permet de partager un ensemble de ressources. Il existe des sémaphores pour la programmation en mémoire partagée. Ici les sémaphore System V sont inter-processus. On définit classiquement deux opérations:

- $P(s,n)$  : « Tester » (de l'allemand *passering* du fait de Dijkstra)
- $V(s,n)$  : « Relâcher » (de l'allemand *vrijgave* du fait de Dijkstra)



# Synchronisation avec des Sémaphores

- $P(s,n)$  : « Tester »
- $V(s,n)$  : « Relâcher »





# Créer des Sémaphores

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semget(key_t key, int nsems, int semflg);
```

- key : Une clef, soit manuelle, soit via ftok ou bien IPC\_PRIVATE
- nsem: nombre de sémaphores à créer
- shmflg: mode de création de la file et ses droits UNIX
  - ➡ IPC\_CREAT crée une file s'il y en a aucune associée à cette clef
  - ➡ IPC\_EXCL échoue s'il existe déjà une file sur la clef indiqué (toujours combiné avec IPC\_CREAT!)
  - ➡ 0600 droit UNIX en octal (important car si omis 0000 et la file et moins pratique !)

# Opération sur des Sémaphores

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semop(int semid, struct sembuf *sops, size_t nsops);
```

- *semid* : identifiant du sémaphore
- *sembuf*: opération(s) à effectuer via un tableau

```
struct sembuf {
    unsigned short sem_num; /* semaphore number */
    short          sem_op;  /* semaphore operation */
    short          sem_flg; /* operation flags */
};
```

➡ *sem\_num*: numéro du sémaphore

➡ *sem\_op*: opération à effectuer

▸ *sem\_op* > 0 : V(s)

▸ *sem\_op* < 0 : P(s)

▸ *sem\_op* == 0 : attente de la valeur 0 -> utile pour synchroniser les processus

➡ Drapeau à utiliser :

▸ *IPC\_NOWAIT*: non-bloquant et renvoie EAGAIN si l'opération avait dû bloquer

▸ *IPC\_UNDO*: demande au noyau d'annuler l'opération si le processus se termine en cas d'arrêt intempestif

- *nsops*: nombre d'opérations à effectuer (elle sont faites de manière atomique)


# Contrôle du Sémaphore

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
int semctl(int semid, int semnum, int cmd, ...);
```

- *semid* : identifiant du sémaphore
- *semnum*: identifiant du sémaphore
- *cmd*: commande à appliquer au sémaphore

**Non défini dans les headers !**



```
union semun {
    int          val; /* Value for SETVAL */
    struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET */
    unsigned short *array; /* Array for GETALL, SETALL */
    struct seminfo *__buf; /* Buffer for IPC_INFO
                           (Linux-specific) */
};
```

➡ IPC\_STAT récupère les informations sur le sémaphore

➡ SETALL définit la valeur du sémaphore (prend un tableau de unsigned short int en paramètre additionnel)

➡ **IPC\_RMID supprime immédiatement le sémaphore et débloque les processus en attente**

➡ ... il existe **BEAUCOUP** d'autres flags voir man

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <sys/wait.h>

int main( int argc, char ** argv ){
    int sem = semget(IPC_PRIVATE, 1, IPC_CREAT | 0600);

    if( sem < 0 ){
        perror("msgget");
        return 1;
    }

    unsigned short val = 1;
    if( semctl(sem, 0, SETALL, &val) < 0 ){
        perror("semctl");
        return 1;
    }

    int pid = fork();
    struct sembuf p;

    p.sem_num = 0;
    p.sem_op = -1;
    p.sem_flg = SEM_UNDO;

    struct sembuf v;

    v.sem_num = 0;
    v.sem_op = 1;
    v.sem_flg = SEM_UNDO;

    if( pid == 0 ) { /* Child */
        while(1){
            if( semop(sem, &p, 1) < 0 ){
                printf("Child: SEM deleted\n");
                return 0;
            }

            printf("CHILD holding the sem\n");
            sleep(1);
            semop(sem, &v, 1);
        }
    }
}

```

## Suite ...

```

    else
    {
        /* Parent */
        int i = 0;
        while(i < 5)
        {
            semop(sem, &p, 1);

            printf("PARENT holding the sem\n");
            sleep(1);
            semop(sem, &v, 1);
            i++;
        }

        /* Parent delete the sem and unlock the child */
        semctl(sem, 0, IPC_RMID);

        wait( NULL );
    }

    return 0;
}

```

# Sortie du Programme

```
$ ./a.out  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
PARENT holding the sem  
CHILD holding the sem  
Child: SEM deleted
```