

Virtualisation et IPCs

M1 - CHPS

Architecture Interne des Systèmes d'exploitations (AISE)

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



Julien Adam
<julien.adam@paratools.com>

Organisation

- Chaque session est découpée en deux parties. Un cours théorique le matin et une mise en pratique l'après-midi (TD) portant sur les connaissances vues le matin.
- Des QCMs sur les bases **importantes** au fil des semaines et portant sur un cours précédent. Le QCM aura toujours lieu durant la matinée
- Une présentation, date de rendu des ressources au **08/01/2023 Minuit**
- Un Examen final le **13/01/2023 (MATIN)**

- 1 - Compilation, Bibliothèques et Layout Mémoire
- 2 - Généralités sur les OS et Entrées-Sorties
- 3 - Mémoire partie 2, Layout Binaire, Runtime
- 4 - Virtualisation et Conteneurs
- 5 - Programmation Noyau
- 6 - Virtualisation et Inter-Process Communications (IPC)
- 7 - Scheduling et Temps-Réel
- 8 - Examen Ecrit et Présentations

Type d'Examen	Coefficient
QCMs	20 %
Présentation	30 %
EXAMEN	50 %

Cours et Corrections



github.com/gweodoo/AISE-23

Programme

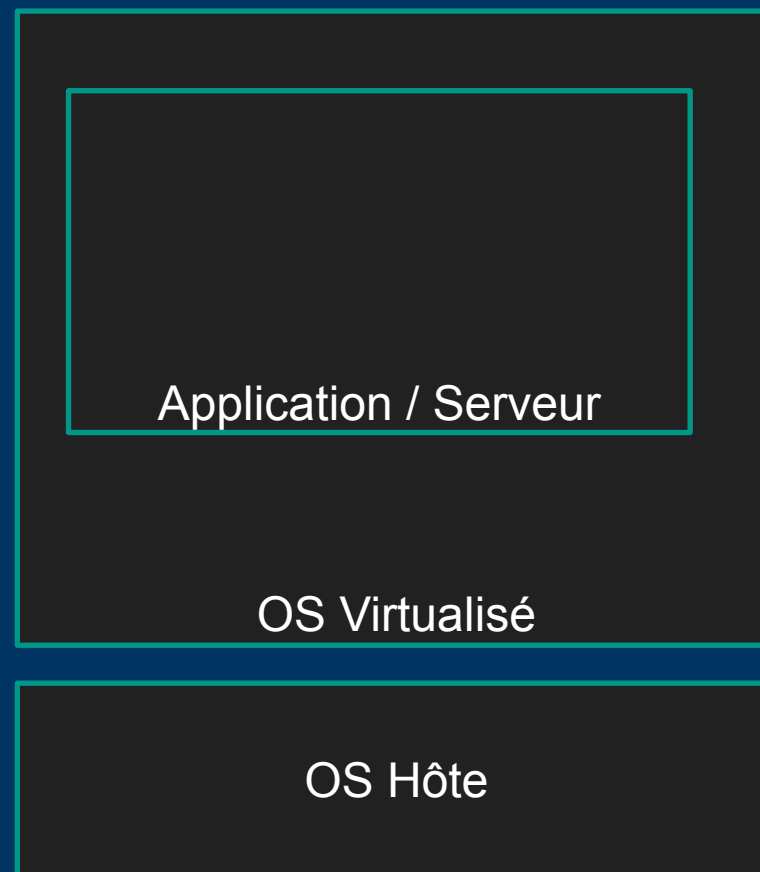
- 1 - Compilation, Bibliothèques et Layout Mémoire
- 2 - Généralités sur les OS et Entrées-Sorties
- 3 - Mémoire partie 2, Layout Binaire, Runtime
- 4 - Virtualisation et Conteneurs
- 5 - Programmation Noyau
- 6 - Virtualisation et Inter-Process Communications (IPC)**
- 7 - Scheduling et Temps-Réel
- 8 - Examen Ecrit et Présentations

Machines Virtuelles

Virtualisation

Une machine virtuelle:

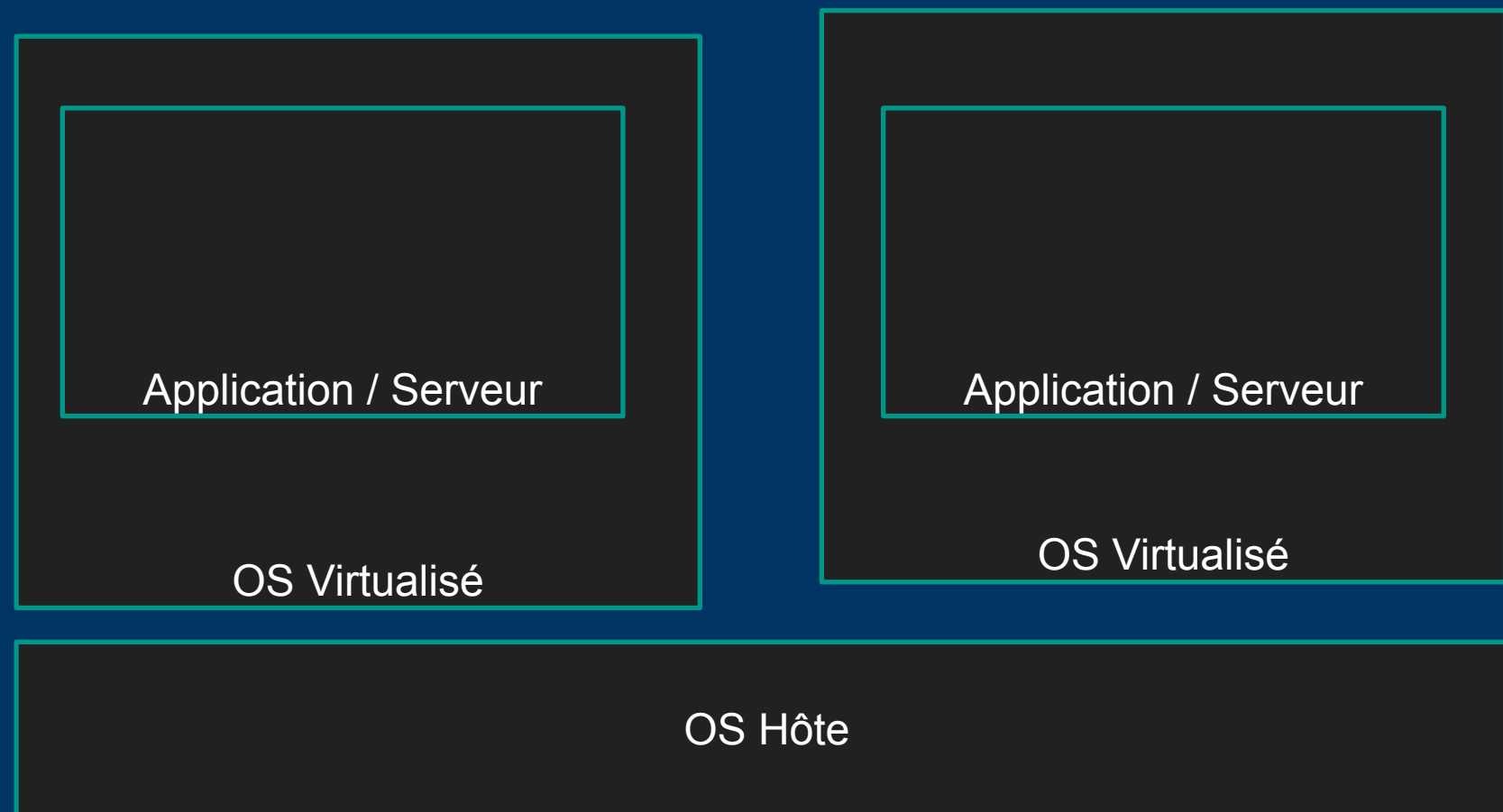
- Émule une machine de manière logicielle pour permettre l'exécution isolée d'un programme;
- Exécute du code dans un contexte spécifique (souvent avec l'aide du matériel) pour le contraindre en terme d'accès;



Virtualisation

Une machine virtuelle:

- Émule une machine de manière logicielle pour permettre l'exécution isolée d'un programme;
- Exécute du code dans un contexte spécifique (souvent avec l'aide du matériel) pour le contraindre en terme d'accès;



Avantages de la Virtualisation

- Regroupement des serveurs sur une même machine. De plus certains serveurs sont faibles en consommation CPU.
- Isolation FORTE des serveurs avec des **systèmes d'exploitation différents et des systèmes de fichiers distincts**;
- **Isolation** y compris vis à vis du matériel (carte réseau virtuelle) et contraintes mémoire cpu explicite (exposition partielle des ressources) — **gestion dynamique des ressources** (CPU Hotplug);
- **Réplication et sauvegarde** facilitée (on sauve l'image disque dans sa totalité), rétablir le système c'est rétablir une image plus récente;
- Facilité d'administration il devient possible de **migrer** un serveur/service donné.

Inconvénients de la Virtualisation

- Les abstractions matérielles ont un coût en performance non négligeable;
- L'OS est totalement répliqué en stockage et en mémoire dans les différentes VMs;
- Si un serveur avec de nombreuses VMs tombe toutes les VMs associées sont inopérantes (besoin de redondance);
- Il y a un overhead d'administration important du fait de la complexité additionnelles des machines séparées.

Votre Première VM

Nous utiliserons qemu installez le !

- (Ubuntu) `sudo apt-get install qemu-kvm qemu virt-manager virt-viewer libvirt-bin`
- (Centos 7) `yum install -y qemu-kvm qemu-img virt-manager libvirt libvirt-python libvirt-client virt-install virt-viewer`
- Votre distrib (go Google)

Créer une image disque de 5GB:

```
qemu-img create -f qcow2 mydebian.qcow2 5G
```

Lancer la VM:

```
qemu-system-XXX [ IMAGE ]
```

qemu-system-aarch64	qemu-system-i386	qemu-system-microblazeel	qemu-system-mipsel	qemu-system-ppc64	qemu-system-sh4eb	qemu-system-unicore32
qemu-system-alpha	qemu-system-lm32	qemu-system-mips	qemu-system-moxie	qemu-system-ppcemb	qemu-system-sparc	qemu-system-x86_64
qemu-system-arm	qemu-system-m68k	qemu-system-mips64	qemu-system-or32	qemu-system-s390x	qemu-system-sparc64	qemu-system-xtensa
qemu-system-cris	qemu-system-microblaze	qemu-system-mips64el	qemu-system-ppc	qemu-system-sh4	qemu-system-tricore	qemu-system-xtensaeb

Votre Première VM

```
Boot failed: could not read the boot disk

Booting from DVD/CD...
Boot failed: Could not read from CDROM (code 0003)
Booting from ROM...
iPXE (PCI 00:03.0) starting execution...ok
iPXE initialising devices...ok

iPXE 1.0.0+git-20161027.b991c67-1 -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI NFS TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 52:54:00:12:34:56 using 82540em on 0000:00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net0 52:54:00:12:34:56)..... ok
net0: 10.0.2.15/255.255.255.0 gw 10.0.2.2
net0: fec0::5054:ff:fe12:3456/64 gw fe80::2
net0: fe80::5054:ff:fe12:3456/64
Nothing to boot: No such file or directory (http://ipxe.org/2d03e13b)
No more network devices

No bootable device.
```

Quel est le problème ??

Votre Première VM

```
Boot failed: could not read the boot disk

Booting from DVD/CD...
Boot failed: Could not read from CDROM (code 0003)
Booting from ROM...
iPXE (PCI 00:03.0) starting execution...ok
iPXE initialising devices...ok

iPXE 1.0.0+git-20161027.b991c67-1 -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI NFS TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 52:54:00:12:34:56 using 82540em on 0000:00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net0 52:54:00:12:34:56)..... ok
net0: 10.0.2.15/255.255.255.0 gw 10.0.2.2
net0: fec0::5054:ff:fe12:3456/64 gw fe80::2
net0: fe80::5054:ff:fe12:3456/64
Nothing to boot: No such file or directory (http://ipxe.org/2d03e13b)
No more network devices

No bootable device.
```

Pas d'OS ! Nous avons passé une image vide !!

Votre Première VM

Démarrer avec un CDROM inséré (comme une vraie machine):

<https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/>

Image d'installation par le réseau de Debian « netinst »:

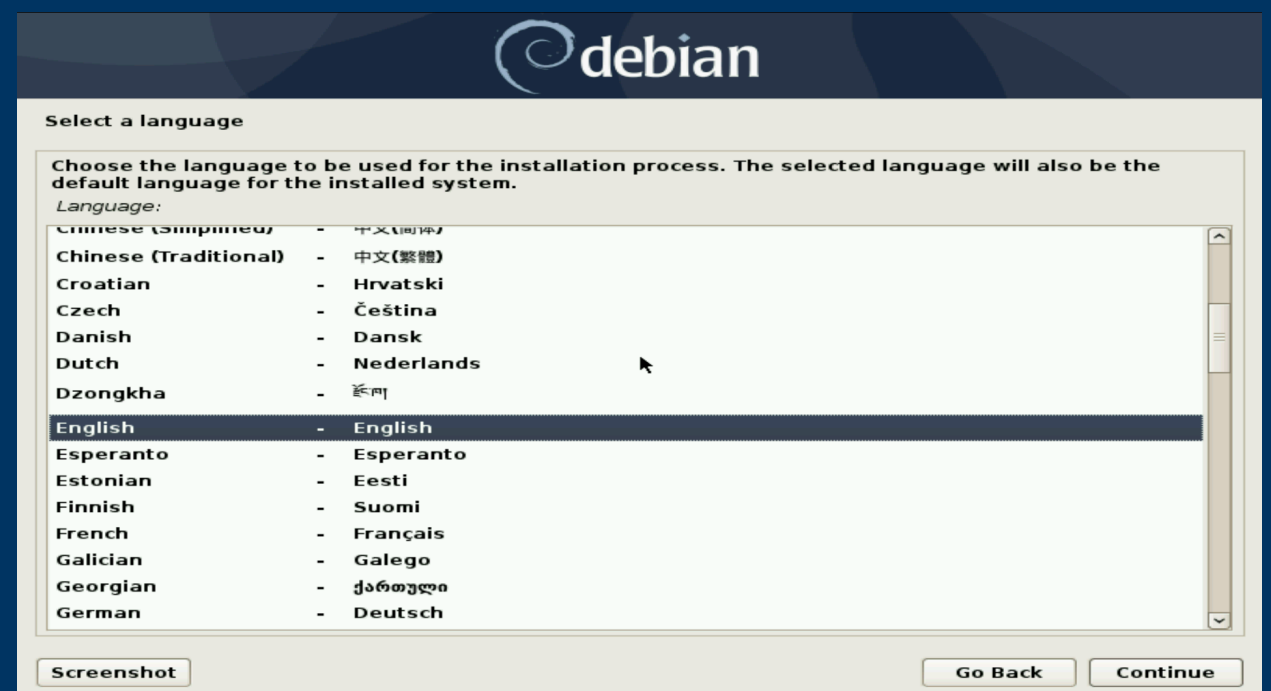
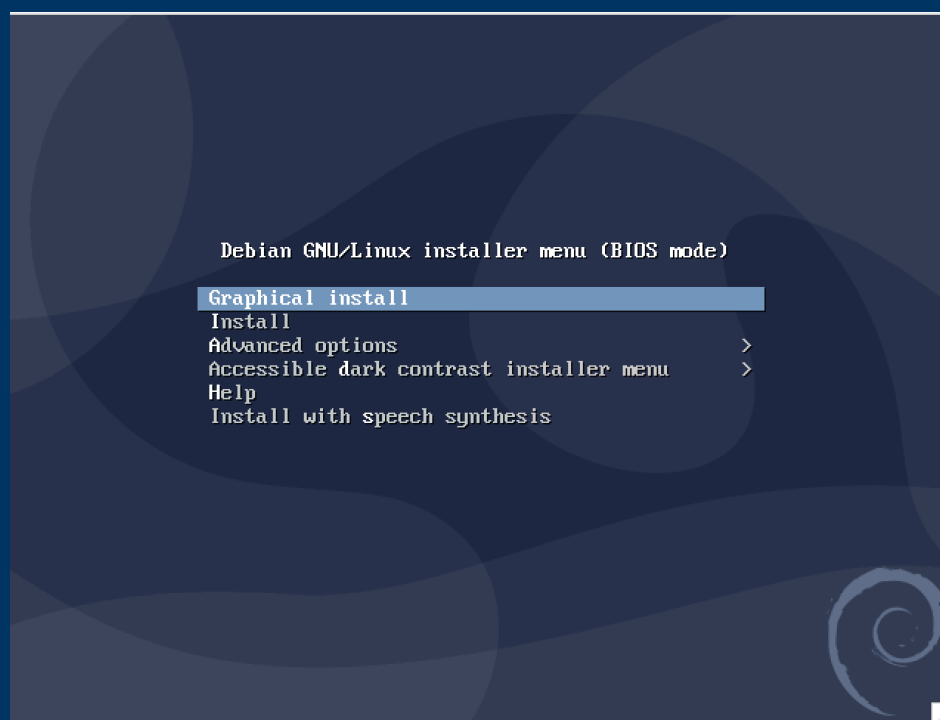
<https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/debian-11.5.0-amd64-netinst.iso>

Téléchargez l'image:

wget <https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/debian-11.5.0-amd64-netinst.iso>

Démarrez la VM avec l'image disque:

```
qemu-system-x86_64 --cdrom ./debian-10.3.0-amd64-netinst.iso -hda mydebian.qcow2 -m 1024  
-netdev user,id=eth0,hostfwd=tcp::10022-:22 -device e1000,netdev=eth0
```



CTRL+ALT / CTRL + ALT + g pour sortir la souris

Image Debian avec Docker

Récupérez une image avec debian:

```
root MDP toto  
chps MDP toto
```

<https://france.paratools.com/chps.qcow2>

CTRL+ALT pour sortir la souris

Votre Première VM

Donner plus de ressources à la VM: `-smp 2 -m 2048`

```
1  [I                                     2.0%]  Tasks: 26, 49 thr; 1 running
2  [                                     0.0%]  Load average: 0.00 0.04 0.14
Mem[|||||]                               197M/1.95G  Uptime: 00:21:09
Swp[                                     0K/0K]
```

Se connecter en ssh:

```
ssh root@localhost -p 10022
```

CTRL+ALT pour sortir la souris

Votre Première VM

Démarrer le système sans affichage: `-nographic`

`CTRL + a puis x` pour quitter

`CTRL + a puis c` pour ouvrir la console qemu (`qemu-monitor`)

Hotplug CPU

```
qemu-system-x86_64 [IMG] -smp 2,maxcpus=4 -nographic
```

```
CTRL + a puis x pour quitter
```

```
CTRL + a puis c pour ouvrir la console qemu (qemu-monitor)
```

```
CTRL + a puis c
```

```
cpu-add 3 (0 et 1 sont déjà présents)
```

Sur l'hôte:

```
$ ssh root@localhost -p 10022 #mdp toto
```

Dans la VM le nouveau CPU n'est pas directement actif:

```
$ echo 1 > /sys/devices/system/cpu/cpu2/online
```

```
1 [ 0.0%]  
2 [ 2.6%]  
Mem[ 133M/1.95G]  
Swp[ 0K/0K]
```

```
1 [ 1.3%]  
2 [ 1.3%]  
3 [ 0.0%]  
Mem[ 133M/1.95G]  
Swp[ 0K/0K]
```

```
1 [ 1.3%]  
2 [ 0.6%]  
3 [ 0.0%]  
4 [ 2.0%]  
Mem[ 136M/1.95G]  
Swp[ 0K/0K]
```

Memory Ballooning

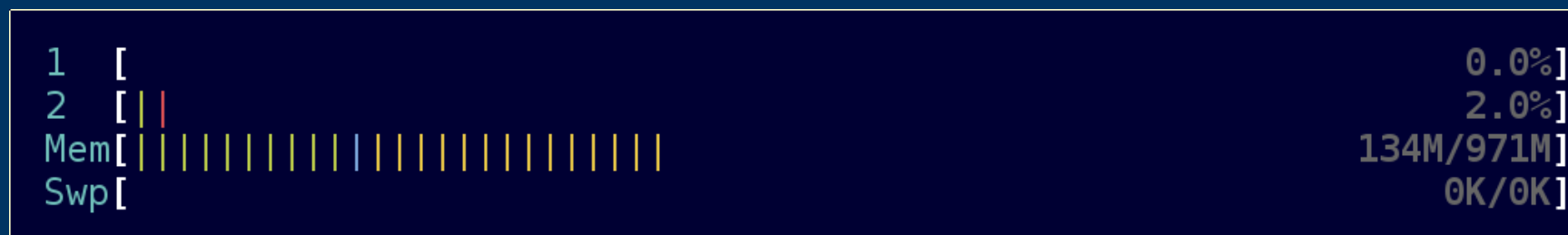
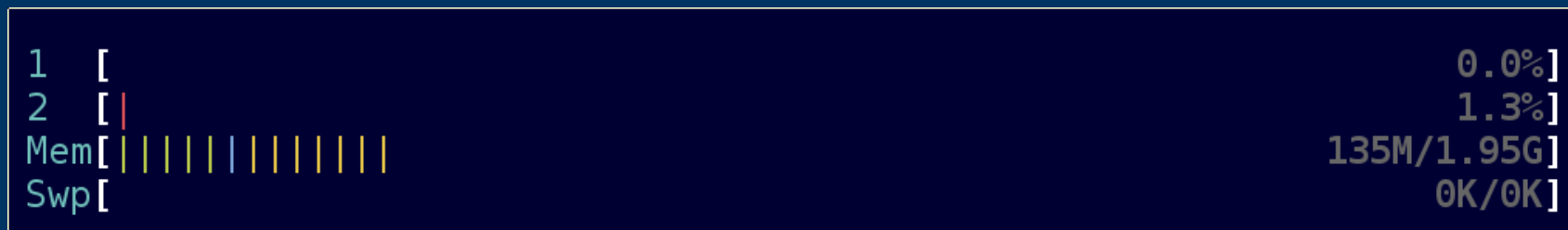
```
qemu-system-x86_64 [IMG] -device virtio-balloon
```

```
# Dépend du chargement des modules virtio dans le kernel cible !
```

```
#CTRL + a puis c pour ouvrir la console qemu (qemu-monitor)
```

```
CTRL + a puis c
```

```
(qemu) balloon 1024
```



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



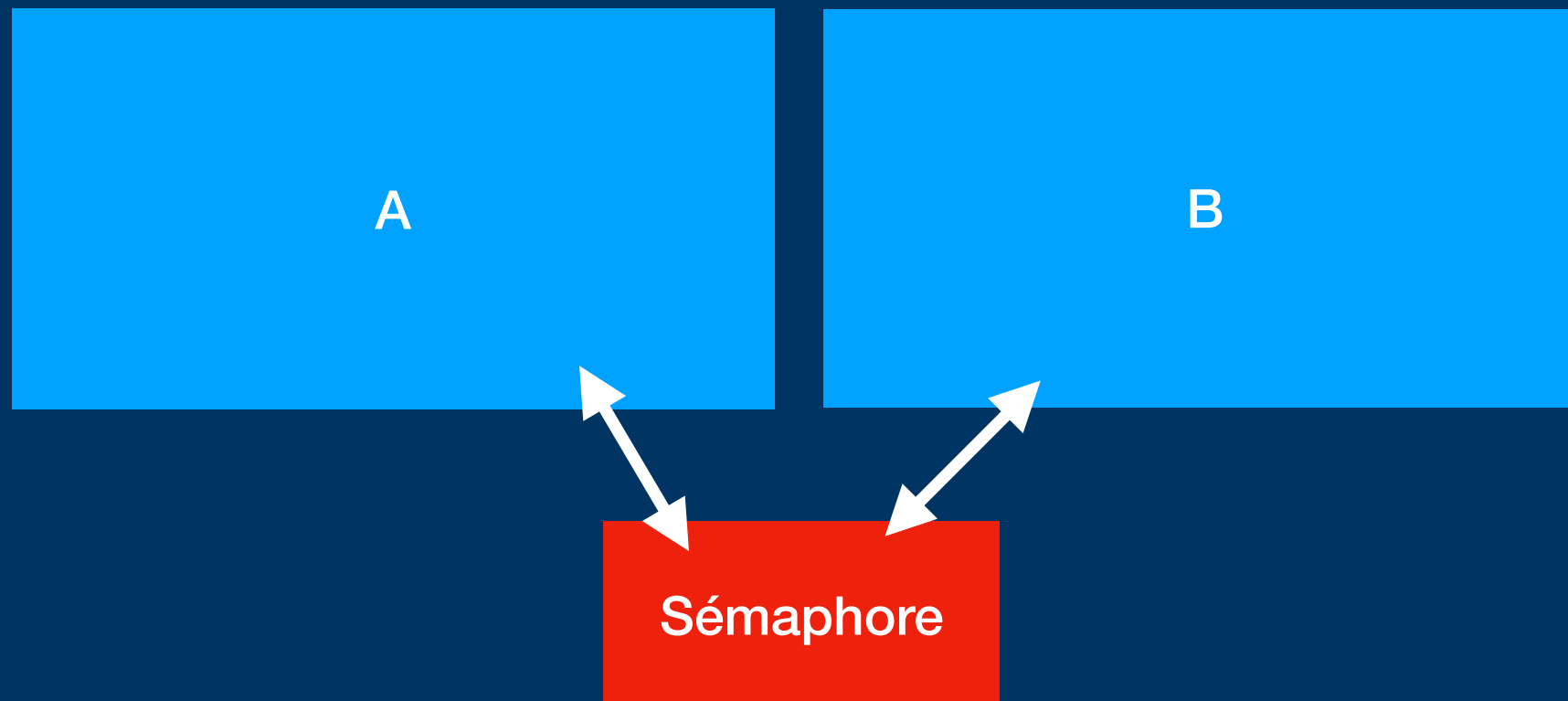
- *ftok*: génération d'une clef IPC
- *msgget*: Récupère un identificateur de file de message
- *msgrecv*: Réception d'un message depuis une file
- *msgsend*: Envoi d'un message dans une file
- *msgctl*: Contrôle de la file de messages

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	jbbesnard	600	1048576	2	dest
0x00000000	39616513	jbbesnard	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 est 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'autres flags voir man

PENSEZ à SUPPRIMER VOS FILES !!!


```

#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
```

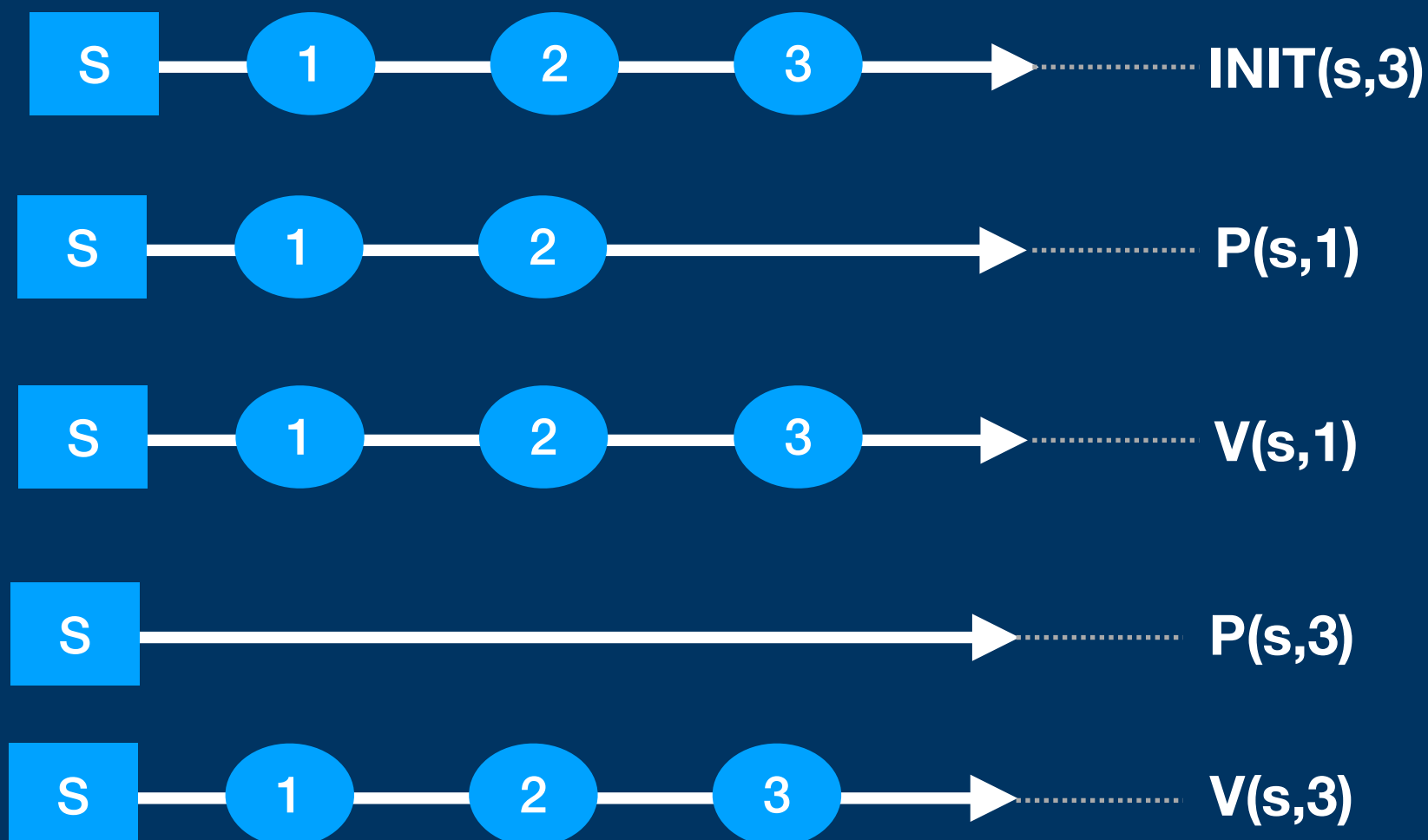
Les Sémaphores

IPC SYSTEM V

Notion de Sémaphore

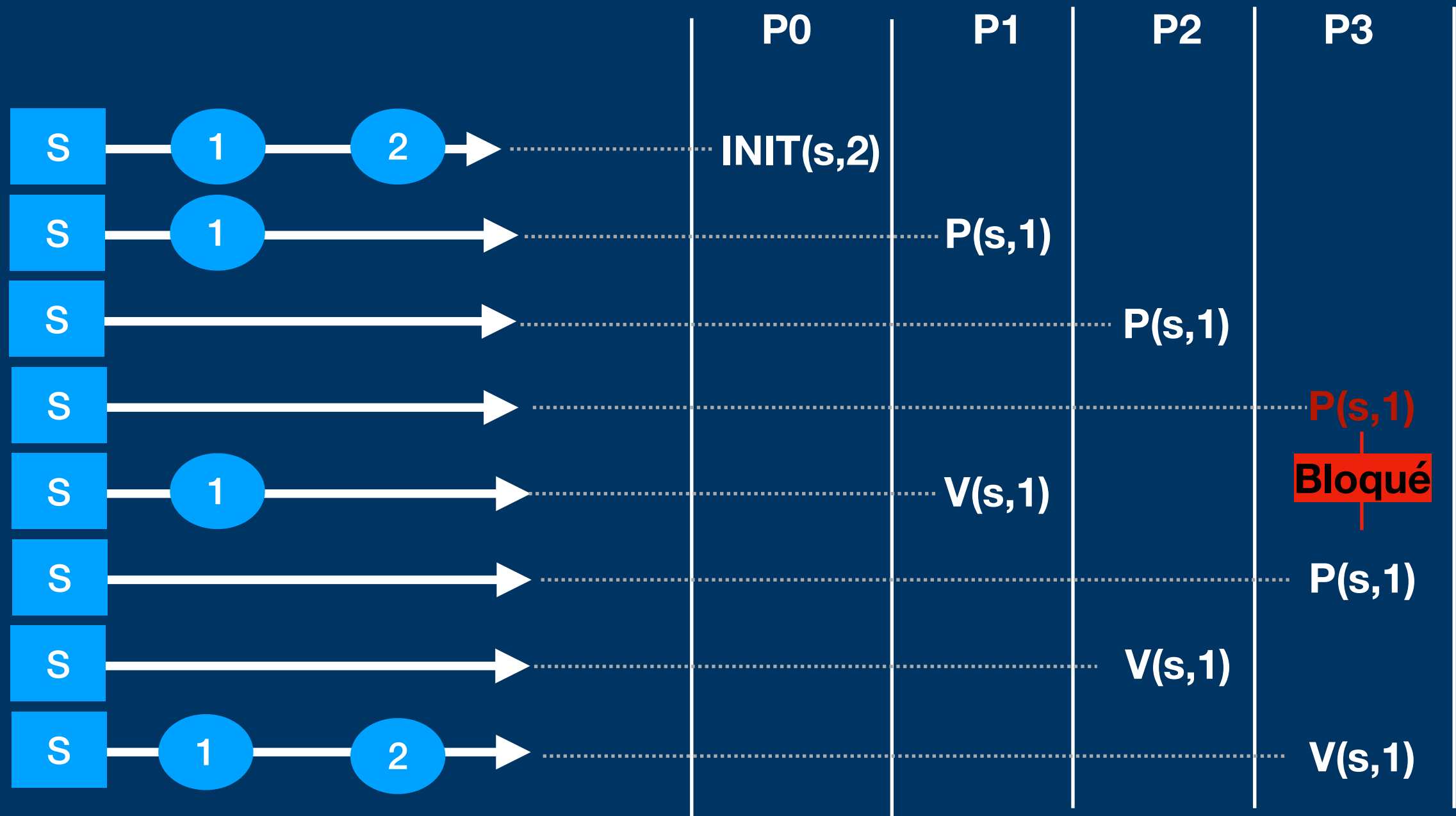
Un sémaphore est un élément de synchronisation qui permet de partager un ensemble de ressources. Il existe des sémaphores pour la programmation en mémoire partagée. Ici les sémaphore System V sont inter-processus. On définit classiquement deux opérations:

- $P(s,n)$: « 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 (elles 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
```

IPCs POSIX

IPC POSIX

Le standard POSIX plus récent propose également les même mécanismes:

- Files de messages
 - Segment de mémoire partagée
 - Sémaphores
-
- Il sont plus fiables en termes de libération et de partage de la ressource;
 - Enfin l'ensemble de l'interface est thread-safe;
 - Les objets sont demandés par nom et non avec une valeur donnée;
 - Ces appels sont un peu moins portable et sont à attendre plus sur des LINUX que des UNIX au sens large;
 - On les décrit généralement comme plus simples à utiliser.

Files de Message POSIX

À vous de jouer avec le man:

- mq_open
- mq_close
- mq_send
- mq_receive
- mq_unlink

Portez l'exemple SYS-V

Que pensez-vous de *mq_notify* ?

Segment SHM POSIX

À vous de jouer avec le man:

- shm_open
- shm_unlink
- mmap

Portez l'exemple SYS-V

Sémaphore IPC POSIX

Aussi « sémaphore nommé » à ne pas confondre avec les sémaphore « anonymes » de la NPTL (libpthread) qui sont dans le même header.

Rappel (ou pas) pour un sémaphore «anonyme »:

- sem_init
- sem_destroy
- **sem_post**
- **sem_wait**

À vous de jouer avec le man pour un sémaphore nommé:

- sem_open
- sem_close
- **sem_post**
- **sem_wait**
- sem_unlink

Portez l'exemple SYS-V

Peut-on l'implémenter avec un sémaphore anonyme et pourquoi ?

**Préférez vous POSIX
ou SYS-V ?**

MMAP

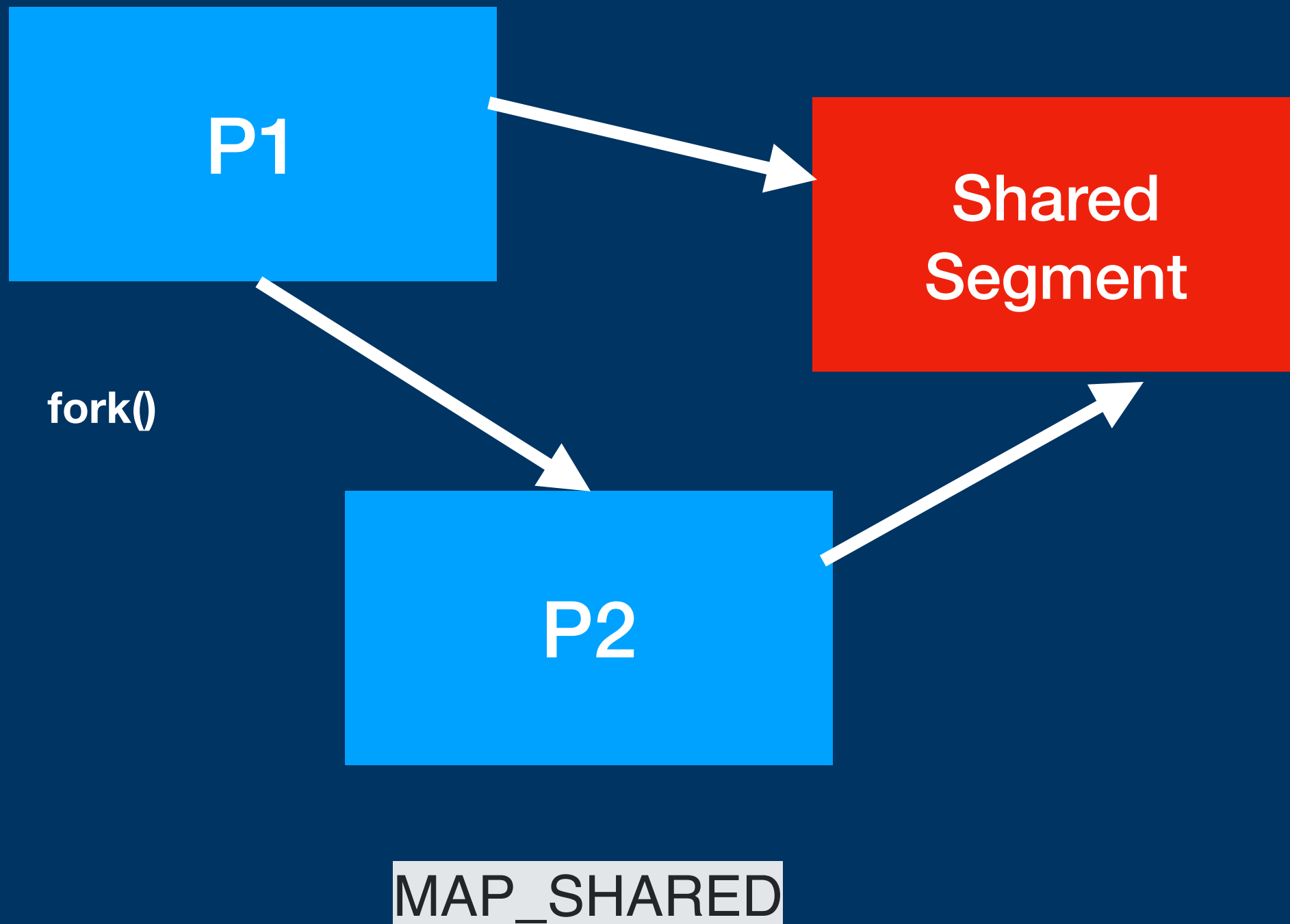
Appel MMAP

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

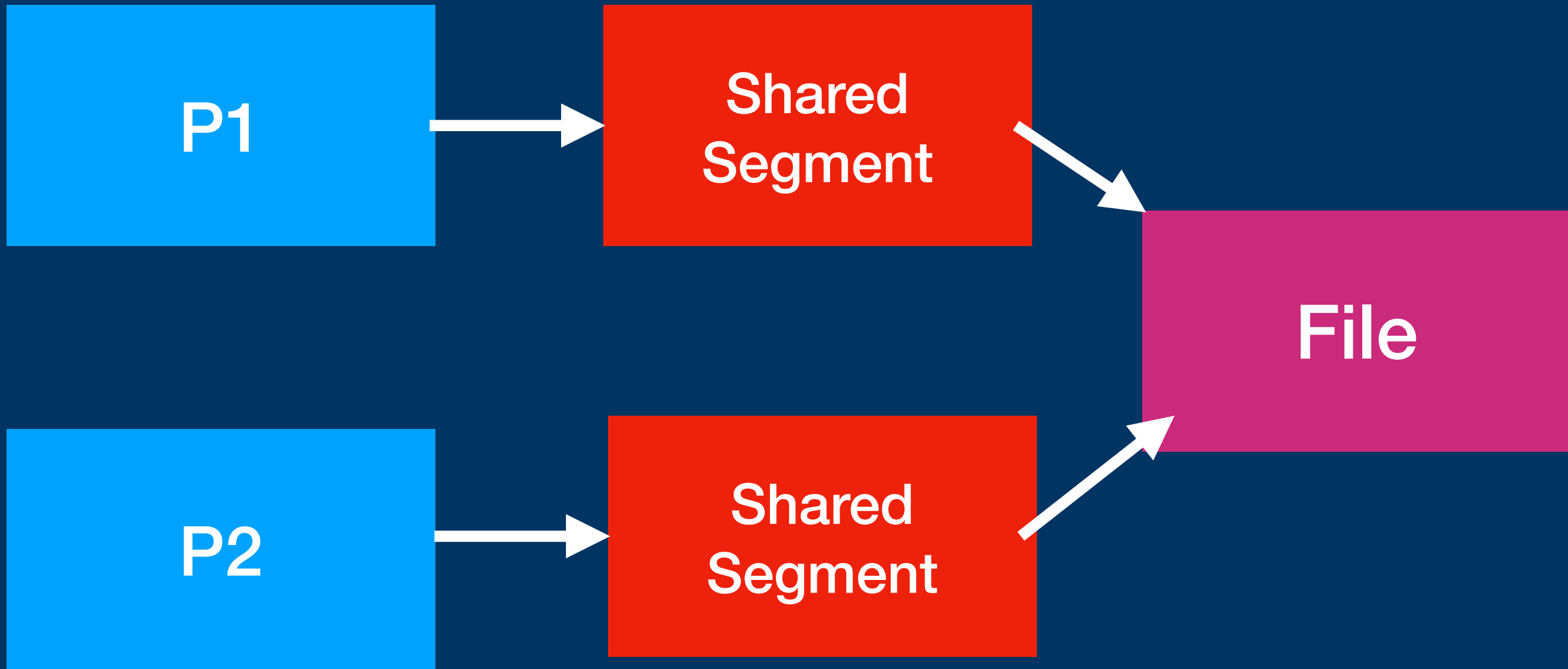
```
void *mmap(void *addr,  
           size_t length,  
           int prot,  
           int flags,  
           int fd,  
           off_t offset);
```

```
int munmap(void *addr, size_t length);
```

Parent to Child



From File



Rappels de Cours (Pipe & Redirections)

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 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000
```


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.

