

## 4. IPC Unix

Outils de communication entre processus sur **la même machine**

Outils principaux:

- Signaux
- Tubes (Pipes)
- IPCs
  - Mémoire partagée (shared memory).
  - File de messages (Message Queue).
  - Sémaphores .

## 4. IPC Unix

Outils principaux:

- Signaux
- Tubes (Pipes)
- IPCs
  - Mémoire partagée (shared memory)
  - File de messages (Message Queue)
  - Sémaphores.

## 4. ICP Unix

### Définition

Un signal est une **interruption logicielle**.

Mécanisme pour informer un processus d'un **événement**.

## 4. ICP Unix

### Signaux

#### Caractéristiques

Un signal peut être généré par le **noyau** ou par d'**autres processus**.

Un signal est délivré à **un** processus spécifique ou à un **groupe** de processus.

## 4. ICP Unix

### Signaux

### Caractéristiques

#### Envoie d'un signal :

1. Le noyau stocke les informations relatives au signal dans une **file d'attente** associée au processus cible.
2. Le noyau envoie un **signal d'interruption** au processus cible pour **interrompre l'opération** en cours et d'exécuter un **gestionnaire de signal**.

#### Réception d'un signal :

Lorsqu'un processus **reçoit un signal** d'un autre processus, il utilise un **gestionnaire de signal** pour **effectuer une action** en réponse au signal.

## 4. ICP Unix

### Signaux

### Limites

Envoi d'une **petite quantité d'informations** entre les processus.

**Non fiables** + **Transmission non garantie** (peuvent être **perdus** ou transmis dans le **désordre**).

# 4. ICP Unix

## Signaux

### Implémentation

Implémenté au niveau du noyau en utilisant une combinaison de **structures de données** et d'**appels système**.

## 4. ICP Unix

### Fonctions ( fichier `signal.h` )

- **SIGKILL** : termine un processus.
- **SIGALRM** : alerte un processus qu'un certain temps s'est écoulé.
- **SIGTERM** : demande à un processus de se terminer.
- **SIGSTOP**: suspens un processus.
- **SIGCONT**: signale à un processus (qui a été arrêté) de continuer.
- **SIGCHLD**: signale à un processus que l'un de ses processus fils s'est terminé.
- **SIGUSR1** et **SIGUSR2** réservés à l'utilisateur. peuvent être utilisés pour n'importe quel but, y compris la communication entre processus..
- ....



## 4. IPC Unix

Outils principaux:

- Signaux
- Tubes (Pipes)
- IPCs
  - Mémoire partagée (shared memory)
  - File de messages (Message Queue)
  - Sémaphores.

## 4. ICP Unix

### Définition

Méthode traditionnelle de communication inter-processus sous UNIX.

Une forme simple d'IPC qui permet à **deux processus liés** de communiquer.

## 4. ICP Unix

## Pipes

### Caractéristiques

Un pipe permet une communication **unidirectionnelle** entre les processus

Un pipe n'a pas d'existence dans un espace de noms de fichiers, il est donc dit **anonyme**.

Il existe un autre type de pipe nommé (appelé FIFO), créé avec l'appel système **mkfifo()** et peut être utilisé entre des processus non liés.

## 4. ICP Unix

### implémentation

Implémenté comme une paire de descripteurs de fichiers dans le noyau.



Fichier pour la **lecture** et fichier pour l'**écriture**.

Un pipe est créé dans le processus père par un appel à **pipe()**.

Les deux descripteurs de fichiers sont reliés par le pipe, qui sert de **tampon** pour les données écrites par un processus et lues par un autre.

## 4. IPC Unix

Outils principaux:

- Signaux
- Tubes (Pipes)
- Prises (Sockets)
- IPCs
  - Mémoire partagée (shared memory).
  - File de messages (Message Queue).
  - Sémaphores .

X

## 4. IPC Unix

### Définition

Mécanismes de communication entre processus **non liés** (non parents) qui se trouvent sur la **même machine**.

### Mécanismes IPC Unix

3 outils :

- ➡ Segment de mémoire partagée 'shared memory'.
- ➡ File de messages 'message queue'.
- ➡ Sémaphores 'semaphore'.

# 4. IPC Unix

## Propriétés

➡ Utilisation d'une IPC:

- Inclure le fichier : `ipc.h`
- Inclure le fichier `msg.h`, `sem.h` ou `shm.h` (selon l'IPC utilisée)



message queue

semaphore

shared memory

## 4. IPC Unix

### Propriétés

- ➔ Identification d'une IPC:
  - **clé**: de type `key.t`.
  - **Identificateur**: `ipcid` donné par le SE lors de la création de l'IPC (avec la fonction `Xget()`).



# 4. IPC Unix

## Propriétés

➔ Identification d'une IPC:

- **clé**: de type `key_t`.
- **Identifiant**: `ipcid` donné par le SE lors de la création de l'IPC (avec la fonction `Xget()`).

choisie aléatoirement par l'utilisateur ou automatiquement par le SE (avec la fonction ftok).

**File TO Key**: fonction système utilisée pour générer une clé IPC à partir du nom d'un fichier.

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

**Exemple.** `ftok(" /c/projet/file.txt", 65);`

## 4. IPC Unix

### Fonctions IPC

3 types de Fonctions pour utiliser les IPC à l'intérieur d'un programme utilisateur:

- ➡ Fonctions **Xget()**.
- ➡ Fonctions **Xctl()**.
- ➡ Fonctions **spécifiques**.

## 4. IPC Unix

### Fonctions IPC

3 types de Fonctions:

➔ Fonctions **Xget()** : **shmget()**, **msgget()**, **semget()**

➔ Fonctions **Xctl()**.

➔ Fonctions spécifiques.

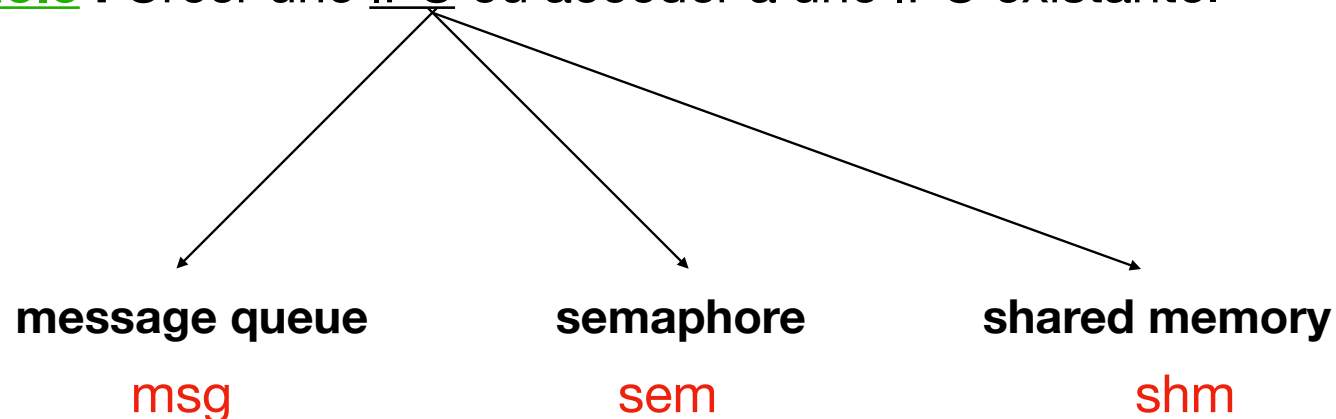
# 4. IPC Unix

## Fonctions IPC

3 types de Fonctions:

➡ Fonctions **X**get() : **shm**get(), **msg**get(), **sem**get()

**Rôle** : Créer une IPC ou accéder à une IPC existante.



## 4. IPC Unix

### Fonctions IPC

3 types de Fonctions:

➡ Fonctions **X**get() : **shm**get(), **msg**get(), **sem**get()

#### Paramètres :

- **key\_t**
- **flag** : de type int (**shm****flg**, **msg****flg**, **sem****flg**)

# 4. IPC Unix

## Fonctions IPC

3 types de Fonctions:

➡ Fonctions **X**get() : **shm**get(), **msg**get(), **sem**get()

### Paramètres :

- **key\_t** ➡ identifier la ressource.
- **flag** : de type int (**shm**flag, **msg**flag, **sem**flag)
  - ➡ Spécifie la **permission** à utiliser lors de la création de l'IPC et des options combinées (par l'opérateur '|' OR)
    - ::: modalités d'ouverture et les droits d'accès

# 4. IPC Unix

## Fonctions IPC

3 types de Fonctions:

➡ Fonctions **X**get() : **shm**get(), **msg**get(), **sem**get()

### Paramètres :

- **key\_t** → identifier la ressource.
- **flag** : de type int (**shm**flg, **msg**flg, **sem**flg)

→ Spécifie la **permission** à utiliser lors de la création de l'IPC et des options combinées (par l'opérateur '|' OR)

::: modalités d'ouverture et les droits d'accès

- **IPC\_CREAT** : créer une nouvelle ressource
- **IPC\_EXCL** : retourner une erreur si la ressource existe déjà
- **IPC\_PRIVATE** : générer une clé automatiquement (par le SE)

## 4. IPC Unix

### Fonctions IPC

- `IPC_CREAT` : créer une nouvelle ressource
- `IPC_EXCL` : retourner une erreur si la ressource existe déjà.
- `IPC_PRIVATE` : générer une clé privée automatiquement

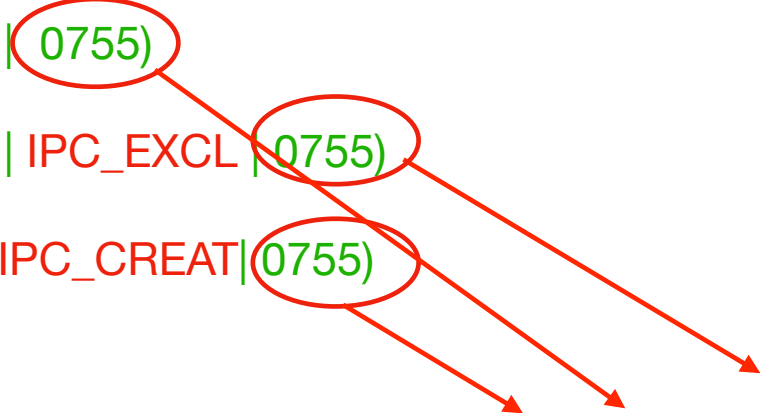
3 types de Fonctions:

➔ Fonctions `Xget()` : `shmget()`, `msgget()`, `semget()`

**Exemple1 :** `msgget (12, IPC_CREAT | 0755)`

**Exemple2 :** `msgget (12, IPC_CREAT | IPC_EXCL | 0755)`

**Exemple3 :** `msgget (IPC_PRIVATE | IPC_CREAT | 0755)`

- 
- `lecteur / écriture` pour le propriétaire
  - `lecture` pour les autres



# 4. IPC Unix

## Fonctions IPC

3 types de Fonctions:

Renvoient un identificateur  
**ipcid**, crée par le SE

➡ Fonctions **Xget()** : **shmget()**, **msgget()**, **semget()**

### Paramètres :

- **key\_t** identifier la ressource.
  - **flag** : de type int (**shmflag**, **msgflag**, **semflag**)
  - 3<sup>ème</sup> paramètre:
    - **Taille** du segment de mémoire partagée (en octets)
    - **Nombre** de l'ensemble de sémaphores
- Exemple:** **semget** (12, **5**, IPC\_PRIVATE)

# 4. IPC Unix

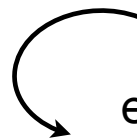
## Fonctions IPC

3 types de Fonctions pour utiliser les IPC à l'intérieur d'un programme utilisateur:

→ Fonctions Xget().

→ Fonctions Xctl().

Rôle : effectuer des **contrôles** sur les IPCs



en utilisant le paramètre **cmd** (commande) dans les descriptifs des fonctions

Exemple : cmd = IPC\_RMID -> **détruire l'IPC**

# 4. IPC Unix

## Fonctions IPC

3 types de Fonctions pour utiliser les IPC à l'intérieur d'un programme utilisateur:

➡ Fonctions Xget().

➡ Fonctions Xctl().

➡ Fonctions **spécifiques**.

- shmat(), shmdet()
- semop()
- msgsnd(), msgrcv()

## 4. IPC Unix

### Fonctions IPC

3 types de Fonctions pour utiliser les IPC à l'intérieur d'un programme utilisateur:

➡ Fonctions Xget().

➡ Fonctions Xctl().

➡ Fonctions **spécifiques**.

- **shmat ()**, **shmdet ()** → **Attachement/ détachement** d'un segment de
- **semop ()** mémoire partagée à l'espace d'adressage d'un
- **msgsnd ()**, **msgrcv ()** processus

# 4. IPC Unix

## Fonctions IPC

3 types de Fonctions pour utiliser les IPC à l'intérieur d'un programme utilisateur:

➡ Fonctions Xget().

➡ Fonctions Xctl().

➡ Fonctions **spécifiques**.

- **shmat ()**, **shmdet ()**

- **semop ()** → Effectuer des **Opérations** sur un ensemble de sémaphores

- **msgsnd ()**, **msgrcv ()**

## 4. IPC Unix

### Fonctions IPC

3 types de Fonctions pour utiliser les IPC à l'intérieur d'un programme utilisateur:

➡ Fonctions Xget().

➡ Fonctions Xctl().

➡ Fonctions **spécifiques**.

- **shmat ()**, **shmdet ()**

- **semop ()**

- **msgsnd ()**, **msgrcv ()** → Envoyer/ recevoir un message par l'intermédiaire d'une file de messages

## 4. IPC Unix

### Commandes Unix

Pour utiliser les IPCs à l'extérieur d'un programme utilisateur, Unix propose 2 commandes:

➡ `ipcs`

➡ `ipcrm`

## 4. IPC Unix

### Commandes Unix

Pour utiliser les IPCs à l'**extérieur** d'un programme utilisateur, Unix propose 2 commandes:

- ➡ **ipcs** : affiche l'état des IPC (**s**: status). Peut-être utiliser avec différentes options : `ipcs -t`, `ipcs -p`, `ipcs -c`, `ipcs -l...`  
<https://man7.org/linux/man-pages/man1/ipcs.1.html>
- ➡ **ipcrm** : détruire une IPC. Syntaxe: `ipcrm [shm | msg | srm] id`

**Exemple** : `ipcrm shm 1`



# 4. IPC Unix

## Commandes Unix

Pour utiliser les IPCs à l'**extérieur** d'un programme utilisateur, Unix propose 2 commandes:

➡ **ipcs** : affiche l'état des IPC (**s**: status). Peut-être utiliser avec différentes options : `ipcs -t`, `ipcs -p`, `ipcs -c`, `ipcs -l...`

<https://man7.org/linux/man-pages/man1/ipcs.1.html>

➡ **ipcrm** : détruire une IPC. Syntaxe: `ipcrm - [m | q | s] id`

```
IPC status from <running system> as of Wed Dec 20 12:52:20 CET 2023
T      ID      KEY      MODE      OWNER      GROUP
Message Queues:

T      ID      KEY      MODE      OWNER      GROUP
Shared Memory:

T      ID      KEY      MODE      OWNER      GROUP
Semaphores:
```

**Exemple** : `ipcrm -s 123`

## 4. IPC Unix

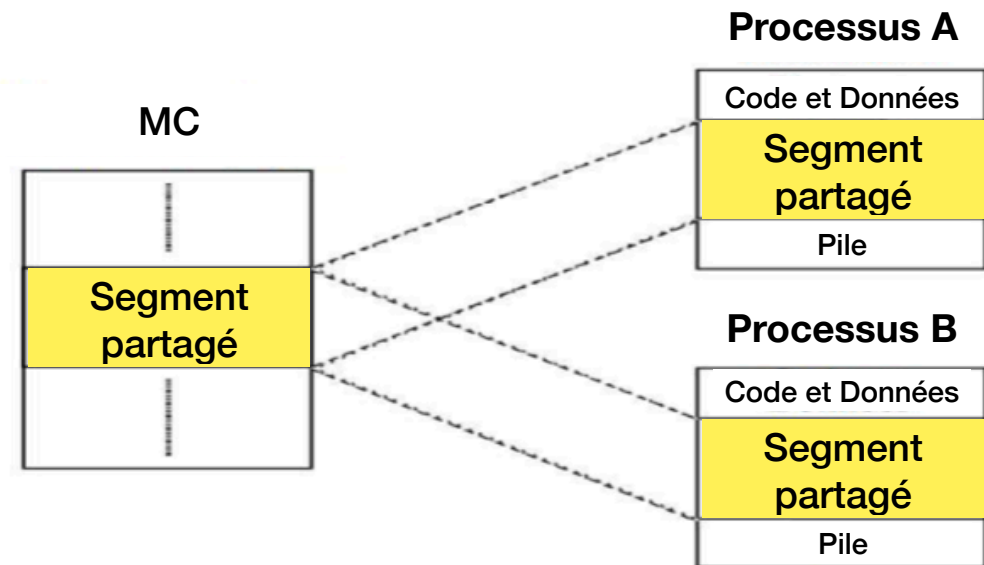
### 1. Segment de mémoire partagée

- Permet à des processus distincts de partager **physiquement** des données.

## 4. IPC Unix

### 1. Segment de mémoire partagée

- Permet à des processus distincts de partager **physiquement** des données.
- Le partage est réalisé par l'**attachement** du segment partagé à l'espace d'adressage de chaque processus.



## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions `<sys/shm.h>`

1 `shm_id = shmget(key, size, flags)`

2 `adr = shmat(shm_id, adr, flags)`

3 `int shmdt(adr)`

4 `int shmctl(shm_id, command, result)`

## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions

1 `shmid = shmget(key, size, flags)`

Créer une zone mémoire partagée, ou d'accéder à une zone déjà existante.

## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions

1 `shmid = shmget (key , size , flags )`

Créer une zone mémoire partagée, ou d'accéder à une zone déjà existante

clé unique permettant  
d'identifier le segment  
de mémoire

modalités  
d'ouverture et les  
droits d'accès

taille du segment de mémoire

## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions

1 `shmid = shmget(key, size, flags)`

Créer une zone mémoire partagée, ou d'accéder à une zone déjà existante

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

<https://man7.org/linux/man-pages/man2/shmget.2.html>

↓  
ipcid

## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions

1 `shmid = shmget(key, size, flags)`

2 `adr = shmat(shmid, adr, flags)`

**attacher** un segment de mémoire partagée à l'espace d'adressage d'un processus.

3 `int shmdt(adr)`

**détacher** un segment de mémoire partagée de l'espace d'adressage d'un processus.



## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions

1 `shmid = shmget(key, size, flags)`

2 `adr = shmat(shmid, adr, flags)`

attacher un segment de mémoire partagée à l'espace d'adressage d'un processus.

3 `int shmdt(adr)`

détacher un segment de mémoire partagée de l'espace d'adressage d'un processus.

identifiant du segment de mémoire  
partagée à attacher

adresse d'attachement/détachement  
(Si adr attachement est NULL, le système choisi  
automatiquement une adr)

indicateurs de contrôle

## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions

1 `shmid = shmget(key, size, flags)`

2 `adr = shmat(shmid, adr, flags)`

3 `int shmdt(adr)`

Adresse physique  
d'attachement ou -1 si erreur

0 si succès et -1 si erreur

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

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

## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions

1 `shmid = shmget(key, size, flags)`

2 `adr = shmat(shmid, adr, flags)`

3 `int shmdt(adr)`

4 `int shmctl (shmid, command, result)`

Effectuer des contrôles sur un segment: consulter , modifier les caractéristiques, supprimer un segment.

## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions

1 `shmid = shmget(key, size, flags)`

commande spécifiant l'opération de contrôle à effectuer.

2 `adr = shmat(shmid, adr, flags)`

3 `int shmdt(adr)`

identifiant du segment de mémoire sur lequel effectuer les contrôles

4 `int shmctl (shmid , command , result )`

adresse pour stocker le résultat (ou récupérer des informations sur le segment de mémoire partagée)

Effectuer des contrôles sur un segment: consulter , modifier ou supprimer un segment.

## 4. IPC Unix

### 1. Segment de mémoire partagée

#### Principales fonctions

1 `shmid = shmget(key, size, flags)`

2 `adr = shmat(shmid, adr, flags)`

3 `int shmdt(adr)`

4 `int shmctl (shmid, command, return_val)`

```
struct shmid_ds {
    struct ipc_perm shm_perm; // Structure pour les autorisations
    size_t shm_segsz;         // Taille du segment de mémoire partagée
    time_t shm_atime;         // Dernier accès (temps en secondes depuis la création)
    time_t shm_dtime;         // Dernière détachement (temps en secondes depuis la création)
    time_t shm_ctime;         // Dernier changement (temps en secondes depuis la création)
    pid_t shm_cpid;           // PID du créateur du segment
    pid_t shm_lpid;           // PID du dernier processus opérant sur le segment
    shmatt_t shm_nattch;      // Nombre d'attachements actuels
    // Autres membres...
};
```

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

0 si succès et -1 si erreur

**Description.** Dans ce programme, le processus père crée un segment de mémoire partagée, écrit x dans ce segment. Ensuite, le processus fils lit cette valeur et l'affiche.

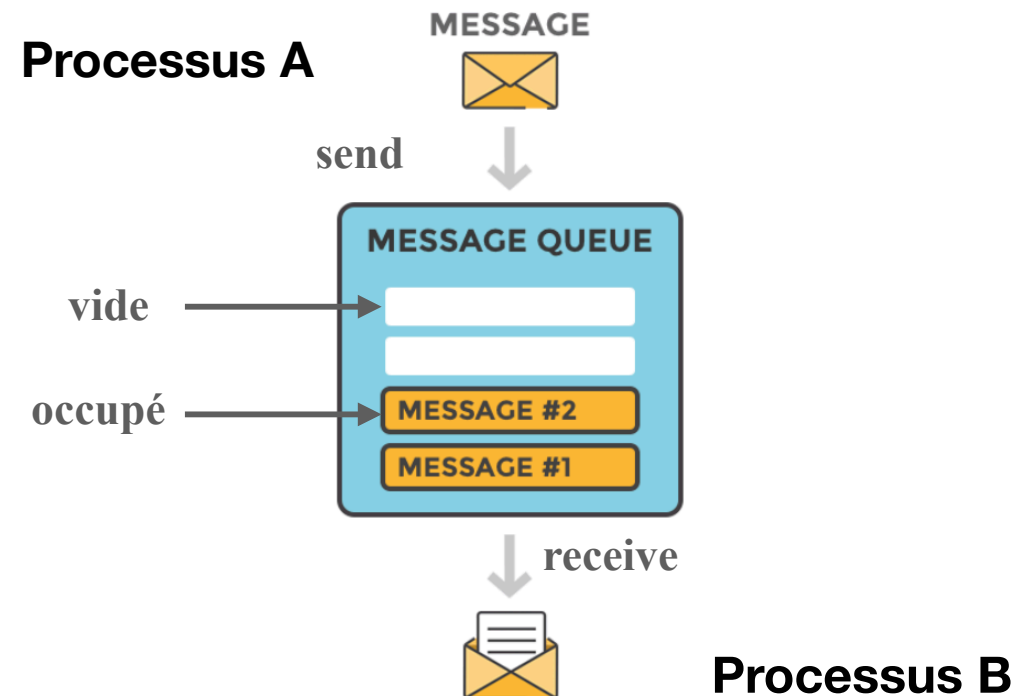
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/ipc.h>
4  #include <sys/shm.h>
5  #include <unistd.h>
6  int main(){
7      int shmid;
8      int *segment;
9      pid_t p1 = fork();
10     if (p1 > 0){                                     // Code du processus père (main)
11         int x = 15;
12         shmid = shmget(2, sizeof(int), 0666 | IPC_CREAT); // créer le segment de mémoire partagée ;
13         segment = (int *)shmat(shmid, NULL, 0);          // attacher le segment de l'espace du père
14         *segment = x;                                    // Déposer X dans le segment
15         wait(NULL);                                     // Attendre p1
16         // code exécuté après la fin de p1
17         shmdt(segment);                                 // détacher le segment
18         shmctl(shmid, IPC_RMID, NULL);}                 // Supprimer le segment
19     else if (p1 == 0){                                  // code du processus fils (p1)
20         shmid = shmget(2, sizeof(int), 0666);           // accéder au segment ayant la clé '2'
21         segment = (int *)shmat(shmid, NULL, 0);         // attacher le segment
22         printf("X = %d\n", *segment);
23         shmdt(segment);                                 // détacher le segment de l'espace de p1
24     return 0;}
```

# 4. IPC Unix

Implantation Unix du concept de **boîte aux lettres**.

Permet à un des processus d'**envoyer** et de **recevoir** des messages à partir d'une même **file d'attente**.

## 2. File de message



## 4. IPC Unix

### 2. File de message

#### Principales fonctions `<sys/msg.h>`

1 `msgid = msgget(key, flags)`

2 `int msgsnd(msgid, msgbuf, size, flags)`

3 `int msgrcv(msgid, msgbuf, maxsize, type, flags)`

4 `int msgctl(msgid, command, buffer)`



# 4. IPC Unix

## 2. File de message

### Principales fonctions

1

```
msgid = msgget(key, flags)
```

Créer une file de messages, ou d'accéder à une file déjà existante

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

<https://man7.org/linux/man-pages/man2/msgget.2.html>

ipcid

# 4. IPC Unix

## 2. File de message

### Principales fonctions

1 `msgid = msgget(key, flags)`

2 `int msgsnd(msgid, msgbuf, size, flags)`

envoyer un message à une file de messages

indicateurs de contrôle  
(Exemple. IPC\_NOWAIT: envoi non bloquant)

identifiant de la file de message

données à envoyer

(pointeur vers la structure de message  
contenant les données à envoyer)

Taille du message en octets  
(taille du bloc de mémoire pointé par  
msgbuf)

# 4. IPC Unix

## 2. File de message

### Principales fonctions

1 `msgid = msgget(key, flags)`

2 `int msgsnd(msgid, msgbuf, size, flags)`

envoyer un message à une file de messages

Signature: `int msgsnd(int msgid, const void msgp[.msgsz], size_t msgsz, int msgflg);`

<https://man7.org/linux/man-pages/man2/msgsnd.2.html>

# 4. IPC Unix

## 2. File de message

### Principales fonctions

1 `msgid = msgget(key, flags)`

2 `int msgsnd(msgid, msgbuf, size, flags)`

3 `int msgrcv ( msgid , msgbuf , maxsize , type , flags )`

Recevoir un message d'une file de messages

identifiant de la file de message

pointeur vers la structure de message qui  
recevra les données

type de message que à recevoir.

indicateurs de contrôle  
(Exemple. IPC\_NOWAIT: réception non  
bloquante)

Taille du message en octets  
(taille du bloc de mémoire pointé par  
msgbuf)

# 4. IPC Unix

## 2. File de message

### Principales fonctions

1 `msgid = msgget(key, flags)`

2 `int msgsnd(msgid, msgbuf, size, flags)`

3 `int msgrcv (msgid, msgbuf, maxsize, type, flags)`

**Recevoir** un message d'une file de messages

**Signature:**

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

# 4. IPC Unix

## 2. File de message

### Principales fonctions

1 `msgid = msgget(key, flags)`

2 `int msgsnd(msgid, msgbuf, size, flags)`

3 `int msgrcv(msgid, msgbuf, maxsize, type, flags)`

4 `int msgctl (msgid, command, buffer)`

Effectuer des contrôles sur une file de message: consulter , modifier ou supprimer une file de message.

# 4. IPC Unix

## 2. File de message

### Principales

1

`msgid_t`

2

`int ms`

3

`int ms`

4

`int msgctl (msgid, command, buffer)`

Signature:

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

```

struct msqid_ds {
    struct ipc_perm msg_perm; // Structure pour les autorisations d'accès
    time_t msg_stime;         // Dernier accès en lecture (en secondes depuis l'époque)
    time_t msg_rtime;         // Dernier accès en écriture (en secondes depuis l'époque)
    time_t msg_ctime;         // Dernière modification (en secondes depuis l'époque)
    unsigned long __msg_cbyte; // Nombre d'octets actuellement dans la file
    msgqnum_t msg_qnum;        // Nombre d'enregistrements dans la file
    msglen_t msg_qbytes;       // Capacité maximale de la file (en octets)
    pid_t msg_lspid;           // PID du dernier processus qui a envoyé un message
    pid_t msg_lrpid;           // PID du dernier processus qui a reçu un message
};

```

**Description.** Dans ce programme, le processus père crée une file de messages, envoie le message 'm' à la file. Ensuite, le processus fils reçoit le message 'm' et l'affiche.

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #include <sys/ipc.h>
5  #include <sys/msg.h>
6  #include <sys/types.h>
7  #include <unistd.h>
8  struct message {
9      long type;
10     char value[100];};
11 int main() {
12     int msgid;
13     struct message m;
14     pid_t p1 = fork();
15     if (p1 > 0){                                     // Code du processus père (main)
16         msgid = msgget(2, 0666 | IPC_CREAT);         // créer la file de message
17         m.type = 1;                                  // assigner m (type)
18         strcpy(m.value, "Bonjour du processus père!"); // assigner m (valeur de m)
19         msgsnd(msgid, &m, sizeof(m.value), 0);       // Envoyer m à la file de messages
20         wait(NULL);                                  // Attendre p1
21         // code exécuté après la fin de p1
22         msgctl(msgid, IPC_RMID, NULL);               // Supprimer la file de messages
23     } else if (p1 == 0){                             // Code du processus fils (p1)
24         msgid = msgget(2, 0666 | IPC_CREAT);         // accéder à la file de message
25         msgrcv(msgid, &m, sizeof(m.value), 1, 0);   // recevoir m de la file de messages
26         printf("Message reçu par le processus fils : %s\n", m.value);} // Afficher m
27     return 0;}

```



## 4. IPC Unix

### 3. Sémaphore

Permet aux processus de **synchroniser** et de **coordonner** leur accès aux ressources partagées.

## 4. IPC Unix

### 3. Sémaphore

Permet aux processus de **synchroniser** et de **coordonner** leur accès aux ressources partagées.

Permet de manipuler un **ensemble** de sémaphores.

Généralement utilisés pour **contrôler la disponibilité** des ressources système telles que les **segments de mémoire partagée**.

## 4. IPC Unix

### 3. Sémaphore

#### Signatures des Principales fonctions `<sys/sem.h>`

<https://man7.org/linux/man-pages/man2/semget.2.html>

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

<https://man7.org/linux/man-pages/man2/semop.2.html>

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

<https://man7.org/linux/man-pages/man2/semctl.2.html>

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

## 4. IPC Unix

### 3. Sémaphore

#### Signatures des Principales fonctions `<sys/sem.h>`

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

Créer un ensemble de sémaphore, ou d'accéder à un ensemble déjà existant

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

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

## 4. IPC Unix

### 3. Sémaphore

#### Signatures des Principales fonctions `<sys/sem.h>`

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

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

Effectuer des **opérations** sur certains sémaphores de l'ensemble

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

## 4. IPC Unix

### 3. Sémaphore

#### Signatures des Principales fonctions `<sys/sem.h>`

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

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

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

Effectuer des opérations de contrôles sur l'ensemble de sémaphore