Développement Système

Inter Processus Communication

Sommaire

- Les ZDC
- Les sémaphores
- Les files de messages

Développement Système: les IPC

Introduction :

Les IPC recouvrent 3 mécanismes de communication entre processus (Inter Processus Communication) :

- Les ***segments de mémoire partagée :*** (**Share Memory**) ou **Zone de Données Commune** (**ZDC**),
- Les ***files de messages*** ou **Q**ueue de **MeS**sages (**MSQ**),
- Les ***sémaphores*** permettent de synchroniser l'accès à des ressources partagées. Ensemble de un ou plusieurs sémaphores,

Développement Système: les IPC

Introduction :

- Les IPC n'appartiennent pas au système de fichier (ils sont donc invisibles avec la commande ls et pas de open, read, write, close),
- Ils n'ont donc pas de numéro d'inode qui assure l'unicité,
- Les IPC permettent de faire communiquer deux processus de manière asynchrone à l'inverse des tubes, c'est-à-dire qu'un processus pourra avoir accès à une information alors que le processus qui a généré cette information est terminé,

Développement Système: les IPC

Introduction :

- Ces outils de communication peuvent être partagés entre des processus n'ayant aucun lien de parenté,
- La mise en œuvre des IPC présente des points communs:
  - ✓ les primitives de création et d'ouverture se présentent sous la forme ***xxxget()***, elles attendent une clef comme l'un des paramètres et retournent un identificateur. La clef est une donnée de type ***key_t*** qui doit être connue de tous les processus utilisant l'IPC,
  - ✓ Les primitives de contrôle se présentent sous la forme ***xxxctl()***,elles permettent d'obtenir ou de modifier des caractéristiques de l'objet et également de détruire l'objet,

Développement Système: les IPC

Introduction :

- Tout objet géré par le noyau est identifié par :
- Une ***identification interne*** correspondant à l'entrée dans une table système et contenant les caractéristiques de l'objet,
- Une ***identification externe*** appelée la ***clef*** utilisée par les processus utilisateurs pour manipuler l'objet

Développement Système: les IPC

Clé :

La clé est obtenue à l'aide de la fonction ***ftok***

key_t ftok(char* pathname, int proj)

A partir d'un nom de fichier existant et accessible, et des huit bits de poids faible de proj (qui doit être non nul), un numéro unique est créé : la clé,

Développement Système: les IPC

Clé : Exemple1

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
key_t clef; /* Clef de l'IPC à générer */
int proj_id;
if ((getuid() & 0xff) != 0)
    proj_id  = getuid();
else proj_id = -1;
/* Génération de la clef (le dernier octet du second argument ne doit pas être à 0) */
if ((clef=ftok(".", proj_id)) == -1)
   perror("ftok");
/* Affichage clef générée */
printf("Clef générée : 0x%04x\n", clef);
printf("Clef générée : %d\n", clef);
}
```

➔ Exécution

Développement Système: les IPC

Clé : Exemple2

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/types.h>

#include <sys/stat.h>
#include <unistd.h>
#include <sys/ipc.h>

int main(int argc, char * argv[])
{
    struct stat status;
    key_t cle1,cle2,cle3;
    if (argc !=2)
    {
        printf("Il faut un argument (nom d'un fichier)\n"); return EXIT_FAILURE;
    }

    if (stat(argv[1],&status) == -1)
    {
        printf("Fichier non existant\n");
        return EXIT_FAILURE;
    }

    cle1 = ftok(argv[1],0);
    cle2 = ftok(argv[1],1);
    printf("Clés créées avec %s :%x (0)\t%x (1)\n",argv[1],cle1,cle2);
    cle3 = ftok(argv[0],0);
    printf("Clé créée avec l'exécutable lui même (%s) (0):%x\n",argv[0],cle3);
    return EXIT_SUCCESS;
}
```

➔ Exécution

Développement Système: les IPC

Commandes shell

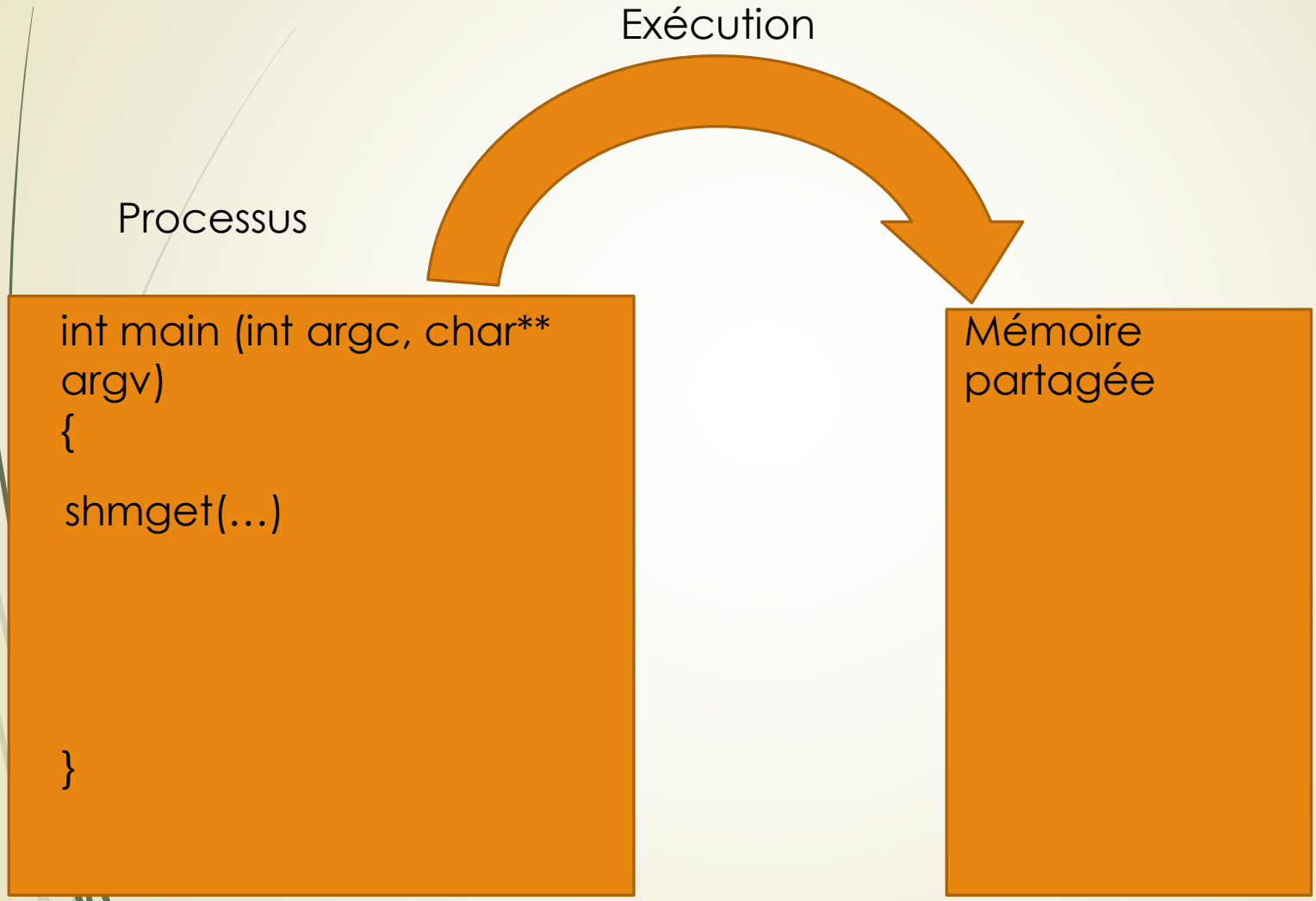
- ***ipcs*** (**IPC Status**): permet d'obtenir toutes les informations sur les IPC,

- ***ipcrm*** (**IPC ReMove**) : permet de supprimer une mémoire partagée, ou un ensemble de sémaphores ou une queue de messages,

Les ZDC

Développement Système: les IPC

ZDC : Création



Développement Système: les IPC

ZDC : Création

- Analyse de la fonction ***shmget*** :

  int shmget(key_t cle, int size, int shmflg);

- **cle** : obtenue avec *ftok*,
- **Size** : la taille de la mémoire partagée,
- **Shmflg** : les options,
  - IPC_CREAT : création,
  - IPC_EXCL : échec si elle existe,
  - 0xxx : droits,

Développement Système: les IPC

ZDC : Exemple3: création

Création Echec si la ZDC existe déjà

Droits

```
#define TAILLE 50
int  main()
{
    //création d'une clé :
    key_t cle = ftok("/etc/passwd",0);
    int shmId = shmget(cle,TAILLE,IPC_CREAT | IPC_EXCL | 0600) ;
    if (shmId == -1)
    {
    //elle existe peut être : essayons de récupérer son identificateur
        shmId = shmget(cle,TAILLE,IPC_EXCL);
        if (shmId == -1)
            {
    //impossible de récupérer l'identifiant
                perror("shmget");
                return -1;
            }
    }
}
```

La ZDC existe déjà donc pas besoin de IPC_CREAT ni de Droits de

Avec cet exemple on a créé une ZDC ou récupéré son identifiant si elle existe déjà

➔ ***Test exemple3***

Développement Système: les IPC

ZDC : Utilisation

Pour utiliser une mémoire partagée il faut :

- **S'attacher à la ZDC** : fonction **shmat**

  char* shmat(int shmid,char* shmaddr,int shmflg);

  -shmid : identificateur de la ZDC (retourné par shmget),

  -shmaddr : si NULL, le système attribue une adresse,

  -shmflg : option **SHM_RND** (lecture seule) ou **0** (lecture/écriture)

- **Écrire dans la ZDC** : **char* pZDC= shmat(„„);**

  ***pZDC ='A';** // pour écrire le caractère A sedans

- **Lire depuis la ZDC** : **char caractere= *pZDC;**

- **Se détacher de la ZDC** : fonction **shmdt**

  int shmdt (char* shmaddr);

  - shmaddr : pointeur retourné par shmat

Développement Système: les IPC

ZDC : contrôle

Pour contrôler une ZDC on utilise la fonction ***shmctl.***

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

- ***shmid :*** identificateur de la ZDC (retourné par shmget)
- ***cmd :*** commande

  - ***IPC_RMID :*** destruction de la ZDC,
  - ***IPC_STAT :*** information sur la ZDC,
  - ***IPC_SET :*** modification des informations de la ZDC,
- ***struct shmid_ds :*** structure d'informations sur la ZDC

cette structure n'est pas nécessaire pour la commande IPC_RMID

```
int shmId = shmget(…);
shmctl(shmId, IPC_RMID, NULL); //destruction
int shmId = shmget(…);
struct shmid_ds info;
shmctl(shmId, IPC_STAT, &info); //lecture des informations
```

Développement Système: les IPC

ZDC : contrôle

Structure shmid_ds

```
struct shmid_ds
{
        struct ipc_perm shm_perm; /* Permissions d'accès */
        int shm_segsz; /* Taille segment en octets */
        time_t shm_atime; /* Heure dernier attachement */
        time_t shm_dtime; /* Heure dernier détachement */
        time_t shm_ctime; /* Heure dernier changement */
        unsigned short shm_cpid; /* PID du créateur */
        unsigned short shm_lpid; /* PID du dernier opérateur */
        short shm_nattch; /* Nombre d'attachements */
};
```

Structure ipc_perm

```
struct ipc_perm
{
        key_t key;
        ushort uid;
        ushort gid;
        ushort cuid;
        ushort cgid;
        ushort mode;
 /* les 9 bits de poids faible représentent
les droits */
        ushort seq;
};
```

Développement Système: les IPC

ZDC : exemple création et infos

Déclarations :

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>

#define Taille 50
int shmDestroy(int shmId);
int shmCreate(char *nom_cle, int taille);
char *shmAttach(int semId);
```

Attachement et détachement :

```c
char * shmAttach(int id)
{
    return shmat(id,NULL,0);
}
```

```c
int shmDestroy(int shmId)
{
    if (shmctl(shmId,IPC_RMID,NULL) == -1)
    {
        perror("shmctl");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

Développement Système: les IPC

ZDC : exemple création et infos

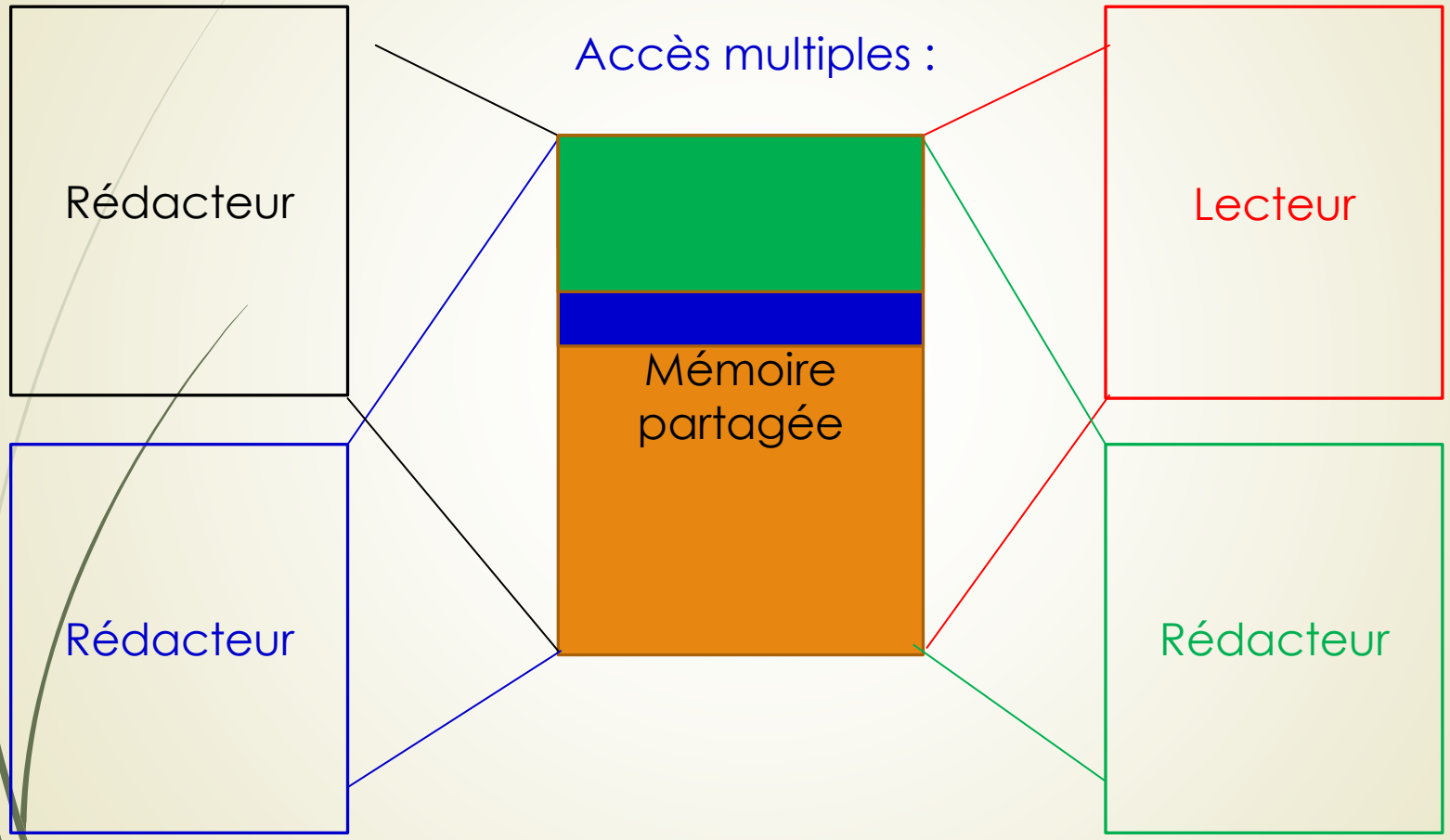
Informations :

```c
int main(int argc, char *argv[])
{
int shmId;
char *pZDC;
struct shmid_ds info;

printf("\n creation \n");
    if ((shmId = shmCreate(argv[0],Taille)) == -1)
    {
        printf("Cette ZDC n'a pas été créée avec l'exécutable CreerZdcEx2 !");
        return EXIT_FAILURE;
    }
    system("ipcs");
    printf("\n infos :\n");
    shmctl(shmId,IPC_STAT,&info);
    printf("\tNombre d'attachement : %ld\n",info.shm_nattch);
    printf("\tTaille : %d\n",info.shm_segsz);
    sleep(3);
    printf("\n attachement\n");
    if ((pZDC = (char *)shmAttach(shmId)) == NULL)
    {
        printf("Impossible de s'attacher !\n");
        return EXIT_FAILURE;
    }
    shmctl(shmId,IPC_STAT,&info);
    printf("\tNombre d'attachement : %ld\n",info.shm_nattch);
    sleep(3);
    system("ipcs");
    sleep(5);
    printf("Destruction\n");
    if(shmDestroy(shmId)==1)
        printf("\n*** Destruction de la ZDC ok ***\n");
    else printf("\t ZDC non detruite\n");
    sleep(3);
    system("ipcs");
    sleep(3);
    shmctl(shmId,IPC_STAT,&info);
    printf("\tNombre d'attachement : %ld\n",info.shm_nattch);
    printf("\n attachement 2 \n");
    if ((pZDC = (char *)shmAttach(shmId)) == NULL)
    {
        printf("Impossible de s'attacher !\n");
        return EXIT_FAILURE;
    }
    sleep(5);
    shmctl(shmId,IPC_STAT,&info);
    printf("\tNombre d'attachement : %ld\n",info.shm_nattch);
    system("ipcs");

    return EXIT_SUCCESS;
}
```

➔ *Exécution*

Développement Système: les IPC

ZDC : Utilisation



Problème : Exclusion mutuelle !!

Les sémaphores

Développement Système: les sémaphores

Les sémaphores : *Introduction*

Si plusieurs processus veulent accéder à une même ressource, des problèmes risquent d'apparaître. Imaginez une mémoire partagée avec plusieurs processus rédacteurs ou un processus rédacteur qui intervient pendant un processus lecteur ! Pour éviter ces problèmes, on utilise des sémaphores qui permettent entre autres l'exclusivité d'une ressource partagée à un seul processus.

Définitions:

- ***Section Critique :*** C'est une partie de code telle que 2 processus ne peuvent s'y trouver au même instant.
- ***Exclusion mutuelle :*** Une ressource est en exclusion mutuelle si seul un processus peut utiliser la ressource à un instant donné.
- ***Conditions de fonctionnement :*** Plusieurs conditions sont nécessaires pour le bon fonctionnement de processus coopérants :

  -Deux processus ne peuvent être, en même temps, en section critique,

  -Aucune hypothèse n'est faite, ni sur la vitesse relative des processus, ni sur le nombre de processeurs,

  -Aucun processus suspendu en dehors d'une section critique ne doit bloquer les autres processus,

  -Aucun processus ne doit attendre "trop longtemps" avant de pouvoir entrer en section critique

Développement Système: les IPC

Les sémaphores : *Définition*

- ***Un sémaphore S** est une variable entière qui n'est accessible qu'au travers de trois opérations Init, P et V.*

● La valeur d'un sémaphore est le nombre d'unités de ressource (exemple : imprimantes...) libres. S'il n'y a qu'une ressource, un sémaphore binaire avec les valeurs 0 ou 1 est utilisé.

● Les opérations doivent être indivisibles (atomiques), ce qui signifie qu'elles ne peuvent pas être exécutées plusieurs fois de manière concurrente. Un processus qui désire exécuter une opération qui est déjà en cours d'exécution par un autre processus doit attendre que le premier termine.

Développement Système: les IPC

Les sémaphores : *les opérations*

- L'opération **Init (S, valeur)** est seulement utilisée pour initialiser le sémaphore S avec une valeur. Cette opération ne doit être réalisée qu'une seule et unique fois.
- L'opération **P(S)** (du néerlandais *Proberen)* signifiant tester et en français "Puis-je ?" ou éventuellement Prise ou Prendre. Cette opération est en attente jusqu'à ce qu'une ressource soit disponible, ressource qui sera immédiatement allouée au processus courant.

SI (S > 0)

ALORS S --;

SINON (attendre sur S)

FSI

Développement Système: les IPC

Les sémaphores : *les opérations*

- L'opération **V(S)** (du néerlandais *Verhogen* signifient tester, incrémenter et en français "Vas- y !" ou éventuellement Vente ou Vendre) est l'opération inverse. Elle rend simplement une ressource disponible à nouveau après que le processus ait terminé de l'utiliser

S++;
SI (des processus sont en attente sur S)

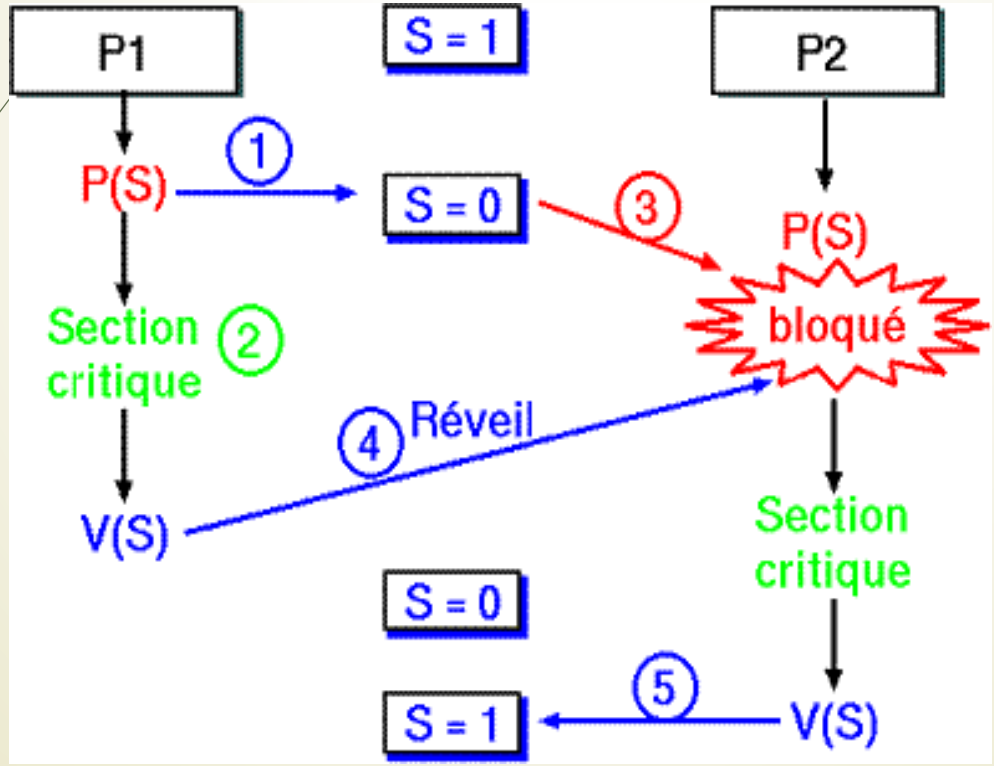
ALORS laisser l'un deux continuer (reveil)

FSI

Développement Système: les IPC

Les sémaphores : Sémaphore binaire

Un sémaphore binaire est un sémaphore qui est initialisé avec la valeur 1. Ceci a pour effet de contrôler l'accès à une ressource unique. Le sémaphore binaire permet l'exclusion mutuelle (mutex) : une ressource est en exclusion mutuelle si un seul processus peut utiliser la ressource à un instant donné



Développement Système: les IPC

Les sémaphores : *Création*

Pour créer un sémaphore on utilise la fonction semget

int semget(key_t clé, int nsems, int semflg);

- ***clé*** : obtenue avec ftok,
- ***nsems*** : le nombre de sémaphores dans l'ensemble,
- ***semflg*** : les options
  - **ICP_CREAT** : création,
  - **IPC_EXCL** : échec si il existe,
  - **0xxx :** droits.

Développement Système: les IPC

Les sémaphores : *Création*

Création

Echec si existe déjà : semget retourne -1

Droits

```
//création d'une clé :
key_t cle = ftok("/etc/passwd",0);
int semId = semget(cle,1, IPC_CREAT | IPC_EXCL | 0600)) ;
if (semId == -1)
{
        //il existe peut être : essayons de récupérer son identificateur
            semId = semget(cle,1, IPC_EXCL);
            if (semId == -1)
            {
        //impossible de récupérer l'identifiant
                    perror("semget");
                    return -1;
            }
}
```

Pas besoin de IPC_CREAT : il existe déjà.
Pas besoin de droits : ils sont déjà fixés.
On récupère l'identificateur.

// on a créé ou récupéré l'identifiant d'un ensemble de 1 sémaphore

Attention *:*

Cet ensemble de sémaphore(s) n'est pas initialisé . . . !

Développement Système: les IPC

Les sémaphores : *Initialisation*

```
key_t cle = ftok("/etc/passwd",0);
int semId = semget(cle,1, IPC_CREAT | IPC_EXCL | 0600)) ;
if (semId == -1)
{
        //il existe peut être : essayons de récupérer son identificateur
        semId = semget(cle,1, IPC_EXCL);
        if (semId == -1)
        {
        //impossible de récupérer l'identifiant
              perror("semget");
              return -1;
        }
}
else
{
        // initialisation avec la valeur 1
        semctl(semId,0,SETVAL,1);
}
```

Ici ? Non : il existe déjà.

Ici ? Non : cas d'erreur.

Ici ? Oui : mais avec un else.

→ ***Exemple1SemCreer , Exemple2Lire_valeur , Exemple3SemInfo***

Développement Système: les IPC

Les sémaphores : *Prendre*

- On prend un sémaphore à l'aide de la fonction : semop

int semop (int semid, struct sembuf *sops, unsigned nsops)

- **semid** : identificateur de l'ensemble (retourné par semget),
- **sops** : adresse d'une structure sembuf,
- **nsops** : nbre d'opérations à exécuter.

struct sembuf :

```
struct sembuf
{
short sem_num; /* Numéro du sémaphore (0=premier) */
short sem_op; /* Opération sur le sémaphore : valeur ajoutée à sem_val*/
short semflg; /*Options pour l'opération */
};
```

Options : **IPC_NOWAIT** (ne pas rester bloqué)

SEM_UNDO (l'opération sera annulée lors de la fin du processus).

Développement Système: les IPC

Les sémaphores : *Prendre*

Soit un ensemble de 1 sémaphore initialisé à la valeur 1.

- **Prendre le sémaphore** :

```
struct sembuf Param;
int semId = semget(…);
Param.sem_num = 0;
Param.sem_op = -1;
Param.sem_flg = SEM_UNDO;
semop(semId, &Param, 1);
```

Autre possibilité d'initialisation :

struct sembuf Param = {0, -1, SEM_UNDO};

Développement Système: les IPC

Les sémaphores : *Vendre*

- **Vendre le sémaphore :**

```
Param.sem_op = 1;
semop(semId, &Param, 1);
```

Les opérations semop sont réalisées **atomiquement** !

Ce qui signifie qu'elles ne peuvent pas être exécutées plusieurs fois de manière concurrente. Un processus qui désire exécuter une opération qui est déjà en cours d'exécution par un autre processus doit attendre que le premier termine.

Développement Système: les IPC

Les sémaphores : *Contrôler*

Pour contrôler un sémaphore on utilise la fonction semctl

int semctl(int *semid*, **int** *semno*, **int** *cmd*, **union semun** *arg* **)**

- **semid** : identificateur de l'ensemble (retourné par semget),
- **semno** : sur quel sémaphore agir,
- **cmd** : commande . . . il y en a 10 !

IPC_RMID, IPC_STAT, IPC_SET, GETALL, GETNCNT, GETPID, GETVAL, GETZCNT, SETALL et SETVAL

- **union semun** : paramètre de cmd.

```
union semun
{
int val;          /* used for SETVAL only */
struct semid_ds *buf;       /* for IPC_STAT and IPC_SET */
ushort *array;            /* used for GETALL and SETALL */
};
```

Développement Système: les IPC

Les sémaphores : *Contrôler*

But	cmd	semno	arg
Destruction	IPC_RMID	S.I.	S.I.
Obtenir des infos	IPC_STAT	S.I.	struct semid_ds *
Modifier des infos	IPC_SET	S.I.	struct semid_ds *
Obtenir toutes les valeurs	GETALL	S.I.	ushort *
Nombre de processus bloqués	GETNCNT	L'indice	S.I.
Processus ayant fait la dernière opération semop	GETPID	S.I.	S.I.
Obtenir la valeur d'un sémaphore	GETVAL	L'indice	S.I.
Nombre de processus en attente de rendez-vous	GETZCNT	L'indice	S.I.
Initialiser tous les sémaphores	SETALL	S.I.	ushort *
Initialiser un sémaphore	SEVAL	L'indice	int

S.I. : **S**ans **I**mportance

Développement Système: les IPC

Les sémaphores : *Exemples*

Créer :

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

int main(int argc, char * argv[])
{
key_t cle;
int semId;

    if (argc != 2)
    {
        printf("Il faut passer la valeur initiale en parametre !\n");
        return EXIT_FAILURE;
    }
    if (atoi(argv[1]) <= 0)
    {
        printf("Il faut une valeur strictement positive\n");
        return EXIT_FAILURE;
    }
    if ((cle = ftok("/etc",0)) == -1)
    {
        perror("ftok");
        return EXIT_FAILURE;
    }
    printf("cle : 0x%X\t",cle);
    if ((semId = semget(cle,1,IPC_CREAT | IPC_EXCL | 0600)) == -1)
    {
        if ((semId = semget(cle,1,IPC_EXCL)) == -1)
        {
            perror("semget");
            return EXIT_FAILURE;
        }
        printf("Le semaphore existait deja j'ai obtenu semId = %d\n",semId);
    }
    else
        printf("semaphore cree avec semId = %d\n",semId);
    if (semctl(semId,0,SETVAL,atoi(argv[1])) == -1)
    {
        perror("semctl SETVAL");
        semctl(semId, 0, IPC_RMID, NULL);
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

Développement Système: les IPC

Les sémaphores : *Exemples*

Infos :

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

int main(int argc, char * argv[])
{
struct semid_ds info;

    if (argc != 2)
    {
        printf("Il me faut un identificateur de semaphore !\n");
        return EXIT_FAILURE;
    }
    if (atoi(argv[1]) < 0)
    {
        printf("Identificateur non valide !\n");
        return EXIT_FAILURE;
    }
    if (semctl(atoi(argv[1]), 29, IPC_STAT, &info) == -1)
    {
        perror("semctl");
        return EXIT_FAILURE;
    }
    printf("Clé : 0x%x\n",info.sem_perm.__key);
    printf("UID propriétaire : %d GID propriétaire : %d\n",info.sem_perm.uid, info.sem_perm.gid);
    printf("droits : %o\n",info.sem_perm.mode & 0777);
    printf("Nombre de sémaphore dans l'ensemble : %ld\n", info.sem_nsems);
    printf("Valeur du semaphore : %d\n",semctl(atoi(argv[1]),0,GETVAL,NULL));
    printf("Nombre de processus etant bloque en attente de ce semaphore : %d\n",semctl(atoi(argv[1]),0,GETNCNT,NULL));
    return EXIT_SUCCESS;
}
```

Développement Système: les IPC

Les sémaphores : *Exemples*

Prendre:

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

int main(int argc, char * argv[])
{
struct sembuf Param;

    if (argc != 2)
    {
        printf("Il me faut un identificateur de semaphore !\n");
        return EXIT_FAILURE;
    }
    if (atoi(argv[1]) < 0)
    {
        printf("Identificateur non valide !\n");
        return EXIT_FAILURE;
    }
    Param.sem_num = 0;
    Param.sem_op = -1;

    if (semop(atoi(argv[1]),&Param,1) == -1)
    {
        perror("semop");
        return EXIT_FAILURE;
    }
    printf(" Attente de semaphore finie\n");

    return EXIT_SUCCESS;
}
```

Développement Système: les IPC

Les sémaphores : *Exemples*

Vendre :

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/sem.h>
#include <unistd.h>

int main(int argc, char * argv[])
{
struct sembuf Param;

    if (argc != 2)
    {
        printf("Il me faut un identificateur de semaphore !\n");
        return EXIT_FAILURE;
    }
    if (atoi(argv[1]) < 0)
    {
        printf("Identificateur non valide !\n");
        return EXIT_FAILURE;
    }
    Param.sem_num = 0;
    Param.sem_op = 1;
    if (semop(atoi(argv[1]),&Param,1) == -1)
    {
        perror("semop");
        return EXIT_FAILURE;
    }
    printf(" semaphore %d libere\n", atoi(argv[1]));
    sleep(1);
    return EXIT_SUCCESS;
}
```

Développement Système: les IPC

Les sémaphores : *Rendez-vous* : ***opération Z***

*Il est possible de créer un rendez-vous de processus, grâce au champ **sem_op** mis à la valeur **0**. Pour cela, on crée un sémaphore initialisé à une valeur correspondant au nombre de processus devant être synchronisés : N. Chaque processus utilisera la fonction **semop** pour prendre le sémaphore puis la fonction **semop** avec la valeur **sem_op** à **0** et lorsque les N processus auront réalisé cette opération, ils seront tous libérés et auront donc été synchronisés sur le plus lent.*

- Permettre de synchroniser N processus.
- On initialise un sémaphore à N.
- Chaque processus prend le sémaphore avec sem_op = -1 : décrémentation de sem_val,

  puis avec sem_op = 0 : attente de sem_val = 0.
- Lorsque le dernier processus prend le sémaphore, il débloque tous les autres.

Développement Système: les IPC

Les sémaphores : *Rendez-vous* : ***opération Z***

Processus 1
semop(-1)
semop(0)
bloqué

semop(i) :
struct sembuf p={0,i,SEM_UNDO};
semop(semId, &p,1);

Processus 2
semop(-1)
semop(0)
bloqué

Sémaphore
sem_val = 0

Processus 3
semop(-1)
semop(0)

Processus 4
semop(-1)
semop(0)
bloqué

Tous les processus sont débloqués.

Développement Système: les IPC

Les sémaphores : *Exemples*

RDV : opération Z

```c
int semZ(int semId)
{
struct sembuf Param;

    Param.sem_num = 0;
    Param.sem_op = -1;
    Param.sem_flg = SEM_UNDO;
    if (semop(semId,&Param,1) == -1)
    {
        perror("semop");
        return EXIT_FAILURE;
    }
    Param.sem_op = 0;
    if (semop(semId,&Param,1) == -1)
    {
        perror("semop");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

RDV : main

```c
int main(int argc, char *argv[]){
int semId;
int i;
pid_t pid[5];
    if (argc != 2)  {
        printf("Il me faut le nombre de processus a creer\n");
        return EXIT_FAILURE;    }
    if ((atoi(argv[1]) < 2) | (atoi(argv[1]) > 5))  {
        printf("La valeur doit etre > 2 et < 5\n");
        return EXIT_FAILURE;    }
    semId = semCreate(argv[0],atoi(argv[1]));
    printf("GETZCNT =  %d\n",semctl(semId,0,GETZCNT,NULL));
    printf("GETVAL = %d\n\n",semctl(semId,0,GETVAL,NULL));
    for(i=0;i<atoi(argv[1]);i++){
        pid[i] = fork();
        switch(pid[i]){
            case -1 : {
                perror("fork");
                semctl(semId,0,IPC_RMID,NULL);
                return EXIT_FAILURE;}
            case 0 : {
                semZ(semId);
                sleep(5);
                printf("Je suis le fils %d de PID : %d je suis debloque\n",i+1,getpid());
                return EXIT_SUCCESS;}
            default : {
                sleep(3);
                printf("GETZCNT =  %d\n",semctl(semId,0,GETZCNT,NULL));
                printf("GETVAL = %d\n\n",semctl(semId,0,GETVAL,NULL));
                sleep(3);}
        }
    }
    printf("attente de terminaison des fils\n");
    for(i=0;i<atoi(argv[1]);i++)    {
        printf("Fils de PID %d termine\n",wait(NULL));  }
    printf("GETZCNT =  %d\n",semctl(semId,0,GETZCNT,NULL));
    printf("GETVAL = %d\n",semctl(semId,0,GETVAL,NULL));
    printf("Destruction du semaphore\n");
    semctl(semId,0,IPC_RMID,NULL);
    return EXIT_SUCCESS;}
```

Les files de messages

Développement Système: les IPC

Les files de messages : introduction

- ***Qu'est-ce qu'un message?***

Un message est une structure dont le premier champ est un entier et le deuxième est une chaine de caractères.

Cette structure ne doit comporter aucune indirection cad aucun pointeur.

Exemple de structure:

```
struct msgbuf
{
        long int mtype;      //type du message, doit être >0
        char mtext[1];       //texte du message
};
```

Développement Système: les IPC

Les files de messages : introduction

- **Qu'est-ce qu'une file de message?**

Une file de messages est une liste chaînée gérée par le noyau dans laquelle un processus peut déposer des données (messages) ou en extraire. Elle correspond au concept de boîte aux lettres.

Un processus peut émettre des messages même si aucun autre processus n'est prêt à les recevoir. Les messages déposés sont conservés après la mort du processus émetteur, jusqu'à leur consommation ou la destruction de la file.

Une propriété essentielle de la file de messages est que le message forme un tout. On le dépose ou on l'extrait en une seule opération.

Développement Système: les IPC

Les files de messages : introduction

- Les avantages principaux de la file de message (par rapport aux tubes et aux tubes nommés) sont :

  - *Un processus peut émettre des messages même si aucun processus n'est prêt à les recevoir*
  - *Les messages déposés sont conservés, même après la mort de l'émetteur, jusqu'à leur consommation ou la destruction de la file.*

- Le principal inconvénient de ce mécanisme est :

  - *la limite de la taille des messages ainsi que celle de la file.*

Développement Système: les IPC

Les files de messages : Création

- *La création d'une file de messages se fait à l'aide de la fonction :*

int msgget (key_t key, int msgflg)

- ***key :*** ***clé obtenue avec ftok,***
- ***msgflg*** **: options**
  - ✓ ***IPC_CREAT*** **: création,**
  - ✓ ***IPC_EXCL*** **: échec si elle existe,**
  - ✓ ***0xxx*** **:droits,**

Développement Système: les IPC

Les files de messages : Création

Echec si existe déjà : msgget retourne -1

Création

Droits

Exemple simple :

```
//création d'une clé :
key_t cle = ftok("/etc/passwd",0);

int msqId = msgget(cle, IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR)) ;
if (msqId == -1)
{
        //elle existe peut être : essayons de récupérer son identificateur
            msqId = msgget(cle, IPC_EXCL);
            if (msqId == -1)
            {
        //impossible de récupérer l'identifiant
                    perror("msgget");
                    return -1;
            }
}
// permet de créer une file de messages ou de récupérer son identifiant
```

Développement Système: les IPC

Les files de messages : Écriture d'un message

- *L'écriture d'un message se fait à l'aide de la fonction msgsnd*

int msgsnd (int msqid, struct msgbuf * msgp, size_t msgsz, int msgflg)

- ***msqid :*** *identificateur de la file de messages (retourné par msgget)*
- ***msgp :*** *adresse du message à écrire,*
- ***msgsz :*** *taille du message (**sans le type**)*
- ***msgflg :*** *option*

  0 : *aucune,*

  IPC_NOWAIT : *ne pas bloquer s'il n'y a pas de place pour le message*

Développement Système: les IPC

Les files de messages : lecture d'un message

- *la lecture d'un message se fait à l'aide de la fonction msgrcv*

ssize msgrcv (int msqid, struct msgbuf * msgp, size_t msgsz, long msgtyp, int msgflg)

- **msqid :** *identificateur de la file de message (retourné par msgget),*
- **msgp :** *adresse où écrire le message lu,*
- **msgsz :** *taille du message (**sans le type**),*
- **msgtyp :** *type du message à lire (1er champ),*

  0 : premier message

  >0 : premier message de ce type

  <0 : premier message dans la file de messages, du plus petit type inférieur ou égal à |type demandé|

- **msgflg :** *option*

  0 : aucune

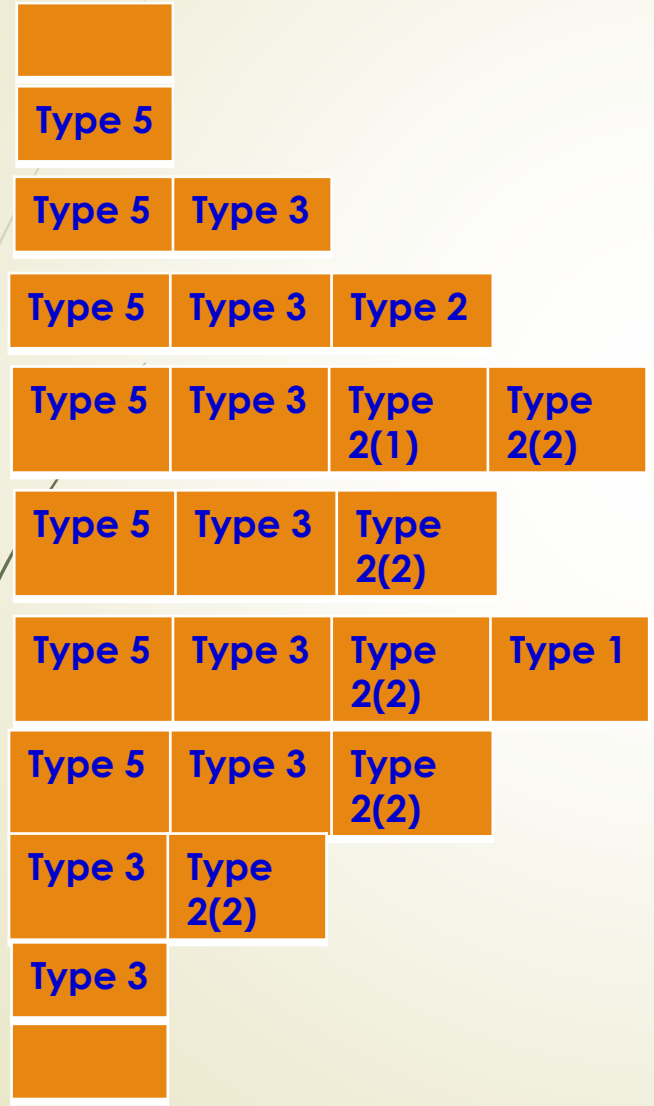
  IPC_NOWAIT : ne pas rester bloqué s'il n'y a pas de message de ce type,

  MSG_NOERROR : tronquer silencieusement les messages trop longs.

La lecture est destructrice !!!

Développement Système: les IPC

Les files de messages : gestion des messages



Écriture d'un message de type 5

Écriture d'un message de type 3

Écriture d'un message de type 2

Écriture d'un message de type 2

Lecture d'un message de type 2

Écriture d'un message de type 1

Lecture d'un message de type -2

Lecture d'un message de type 0

Lecture d'un message de type 2

Lecture d'un message de type -10

Développement Système: les IPC

Les files de messages : Contrôler une file de messages

- *Le contrôle d'une file de message se fait à l'aide de la fonction msgctl*

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

- **msqid :** *identificateur de la file de messages(retourné par msgget),*
- **cmd :** *commande*

  IPC_RMID : destruction de la file de messages,

  IPC_STAT : informations sur la file de messages,

  IPC_SET : modifier les informations de la file de messages,

- **struct msqid_ds :** *structure d'informations sur la file de messages (pas nécessaire pour la commande IPC_RMID).*

Développement Système: les IPC

Les files de messages : Contrôler une file de messages

- ***Strucrture msqid_ds :***

```
struct msqid_ds
{
   struct ipc_perm msg_perm;     /* Propriétaire et permissions */
   time_t  msg_stime;    /* Heure du dernier msgsnd(2) */
   time_t  msg_rtime;    /* Heure du dernier msgrcv(2) */
   time_t  msg_ctime;    /* Heure de dernière modification */
   unsigned long   __msg_cbytes; /* Nombre actuel d'octets dans la file(non standard) */
   msgqnum_t   msg_qnum;     /* Nombre actuel de messages dans la file */
   msglen_t    msg_qbytes;   /* Nombre maximal d'octets autorisés dans la file */
   pid_t   msg_lspid;    /* PID du dernier msgsnd(2) */
   pid_t   msg_lrpid;    /* PID du dernier msgrcv(2) */
};
```

Développement Système: les IPC

Les files de messages : *Exemples*

- ***Exemple 1:*** *Création d'une file de messages et envoi de message,*

```c
#define TAILLE 20
typedef struct{
    long mtype;
    char Message[TAILLE];
}MSGRQ;
int main (int argc, char *argv[]){
    key_t cle;
    int msgId;
    MSGRQ requete;
    cle = ftok("/etc/passwd",0);
    if (cle == -1)  {
        perror("ftok");
        return EXIT_FAILURE;
    }
    msgId = msgget(cle,IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    if (msgId == -1){
        msgId = msgget(cle,IPC_EXCL);
        if (msgId == -1){
            perror("msgget");
            return EXIT_FAILURE;
        }
        printf("La MSG existait deja identificateur : %d\n",msgId);
    }
    if (argc != 1)
        requete.mtype = atoi(argv[1]);
    else
        requete.mtype = 1;
    if (argc > 2)
        strcpy(requete.Message, argv[2]);
    else
        strcpy(requete.Message,"Message generique");
    if ((msgsnd(msgId,&requete,sizeof(MSGRQ)-sizeof(long),0)) == -1){
        perror("msgsnd");
        return EXIT_FAILURE;
    }
    printf("MSG creee avec la cle %x\n",cle);
    printf("Identificateur de la MSG : %d\n",msgId);
    printf("\n message depose : %s    de type %d\n", requete.Message,requete.mtype);
    return EXIT_SUCCESS;
}
```

Développement Système: les IPC

Les files de messages : *Exemples*

- ***Exemple2*** : *Lecture de message*

```c
#define TAILLE 20
typedef struct
{
    long mtype;
    char Message[TAILLE];
}MSGRQ;
int main (int argc, char *argv[]){
    MSGRQ ReqRecu;
    key_t cle;
    int msgId;
    if (argc != 2)  {
        printf("Il faut 1 parametres : le type de message a lire\n");
        return EXIT_FAILURE;
    }
    cle = ftok("/etc/passwd",0);
    if (cle == -1)  {
        perror("ftok");
        return EXIT_FAILURE;
    }
    msgId = msgget(cle,IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    if (msgId == -1)    {
        msgId = msgget(cle,IPC_EXCL);
        if (msgId == -1)    {
            perror("msgget");
            return EXIT_FAILURE;
        }
        printf("La MSG existait deja identificateur : %d\n",msgId);
    }
    if ((msgrcv(msgId,&ReqRecu,sizeof(MSGRQ)-sizeof(long),atoi(argv[1]),0)) == -1){
        perror("msgrcv");
        return EXIT_FAILURE;
    }
    printf("Lecture des messages de type %ld sur la MSG d'identificateur %d :\n\n",atoi(argv[1]),msgId);
    printf("type : %ld\n",ReqRecu.mtype);
    printf("Message : %s\n",ReqRecu.Message);
    return EXIT_SUCCESS;
}
```

Développement Système: les IPC

Les files de messages : *Exemples*

- ***Exemple 3 :*** *Informations sur une file de messages,*

```c
int main (int argc, char *argv[])
{
    int msgId;
    struct msqid_ds info;

    if (argc != 2)  {
        printf("Il faut 1 parametre : identificateur  de la msg dont on veut les infos\n");
        return EXIT_FAILURE;
    }
    msgId= atoi(argv[1]);
    if (msgctl(msgId,IPC_STAT,&info) == -1) {
        perror("msgctl");
        return EXIT_FAILURE;
    }
    printf("uid = %d gid = %d droits = %o\n",info.msg_perm.uid, info.msg_perm.gid, info.msg_perm.mode);
    printf("Nombre de messages restants : %ld\n",info.msg_qnum);
    printf("Taille de tous les messages : %ld\n",info.__msg_cbytes);
    printf("Taille de la file de message : %ld\n",info.msg_qbytes);
    printf("Taille restante de la file de message : %ld\n",info.msg_qbytes- info.__msg_cbytes);

    return EXIT_SUCCESS;
}
```

Développement Système: les IPC

Les files de messages : *Exemples*

Exemple 4 : *Modification de la taille d'une file de messages (limitation à 100 octets)*

```c
int main (int argc, char *argv[]){
    key_t cle;
    int msgId;
    MSGRQ requete;
    struct msqid_ds info;
    cle = ftok("/etc/passwd",0);
    if (cle == -1)  {
    msgId = msgget(cle,IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    if (msgId == -1)    {
    if (msgctl(msgId, IPC_STAT, &info) == -1)   {
        perror("msgctl1");
        msgctl(msgId, IPC_RMID, NULL);
        return -1;
    }
    if (info.msg_qbytes > 100)  {
        info.msg_qbytes = 100;
        if (msgctl(msgId, IPC_SET, &info) == -1)    {
            perror("msgctl2");
            msgctl(msgId, IPC_RMID, NULL);
            return -1;
        }
    }
    if (argc != 1)
        requete.mtype = atoi(argv[1]);
    else
        requete.mtype = 1;
    strcpy(requete.Message,"Un message de type1");
    if ((msgsnd(msgId,&requete,sizeof(MSGRQ)-sizeof(long),0)) == -1){
        perror("msgsnd");
        return EXIT_FAILURE;
    }
    printf("MSG creee avec la cle %x\n",cle);
    printf("Identificateur de la MSG : %d\n",msgId);
    if (msgctl(msgId, IPC_STAT, &info) == -1)   {
        perror("msgctl3");
        msgctl(msgId, IPC_RMID, NULL);
        return -1;
    }
    printf("Taille de la file de message : %ld\n",info.msg_qbytes);
    printf("Taille restante de la file de message : %ld\n",info.msg_qbytes- info.__msg_cbytes);
    return EXIT_SUCCESS;
}
```