Programmation Système

Les IPC SYSTEM V

Université de Tours
Faculté des Sciences et Techniques
Antenne Universitaire de Blois

Licence Sciences et Technologies
Mention : Informatique
2ème Année

Mohamed TAGHELIT
taghelit@univ-tours.fr

Les IPC SYSTEM V

Inter **P**rocess **C**ommunication (System V)

- Communications et synchronisations entre processus **locaux**
- Trois mécanismes
  - La mémoire partagée
  - Les files de messages
  - Les sémaphores

Caractéristiques Communes aux IPC

- Une table par mécanisme

  une entrée → une instance
- Une clé numérique par entrée
- Un appel système *xxx*`get` par mécanisme

  xxx → `shm` (mémoire partagée), `msg` (files de messages) ou `sem` (sémaphores)
  - créer une nouvelle entrée
  - retrouver une déjà existante
  - retourne un descripteur

  Cas de création
  - `cle = IPC_PRIVATE`
  - `IPC_CREAT` → `flags`
  - `IPC_CREAT | IPC_EXCL` → `flags`

Structure Commune aux IPC

- Une structure commune

```
struct  ipc_perm  {
   ushort   uid;          /* owner's user id */
   ushort   gid;          /* owner's group id */
   ushort   cuid;         /* creator's user id */
   ushort   cgid;         /* creator's group id */
   ushort   mode;         /* access modes */
   ushort   seq;          /* slot usage sequence number */
   key_t    key;          /* key */
   };
```

Définitions Communes aux IPC

- Définitions communes

```
#define IPC_CREAT   0001000     /* create entry if key doesn't exist */
#define IPC_EXCL    0002000     /* fail if key exists */
#define IPC_NOWAIT  0004000     /* error if request must wait */

#define IPC_PRIVATE (key_t)0    /* private key */

#define IPC_RMID    0           /* remove identifier */
#define IPC_SET     1           /* set options */
#define IPC_STAT    2           /* get options */
```

Composition d'une Clé

- Soumission d'une clé pour obtenir un descripteur d'IPC

- Composition d'une clé
  - Fixée par l'utilisateur
  - Déterminée par le système

```
key_t ftok(path, code);
const char *path;
int code;
```

`path` doit exister tant que des clés y sont associées.

```
[student]$ touch file
[student]$ ls -i file
263139 file
[student]$ \rm file
[student]$ touch titi
[student]$ touch file
[student]$ ls -i file
267476 file
[student]$
```

Les commandes `Shell` Associées

Deux commandes `shell`

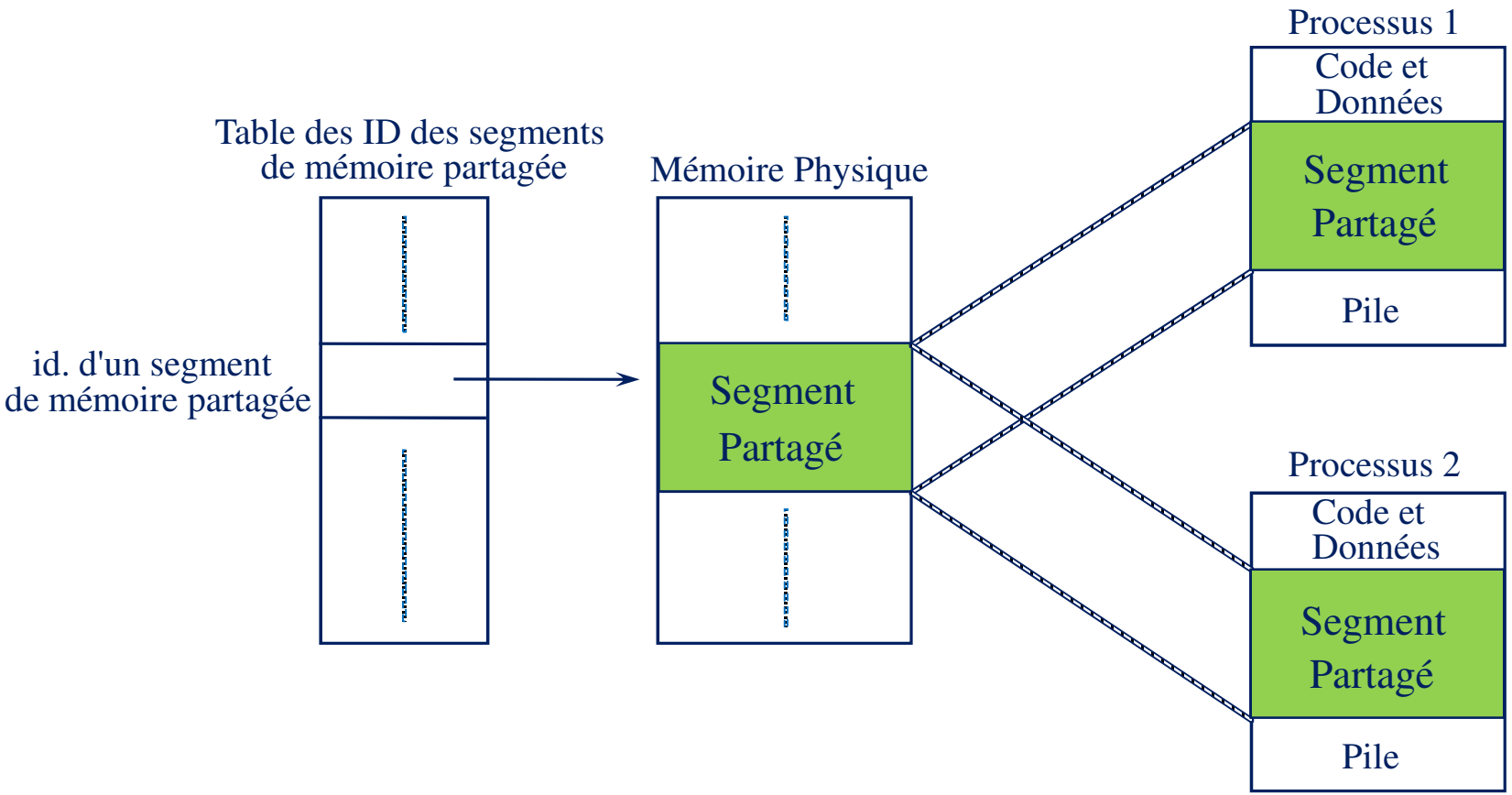
- `ipcs`
  - liste des ressources actives ainsi que leurs caractéristiques

```
$ipcs
IPC        ID     KEY          MODE         OWNER        GROUP
Messages Queues:
q          0      0x00000000   --rw-------  root         root
Shared Memory:
m          3      0x41442041   --rw-rw-rw   root         root
Semaphores:
s          1      0x4144314d   --rw-rw-rw-  root         root
```

- `ipcrm`
  - suppression des ressources

```
$ipcrm -q 0 -m 3 -S 0x4144314d
```

Segments de Mémoire Partagée



Zone mémoire attachée à un processus mais qui peut être accédée par d'autres processus

Création d'un Segment de Mémoire Partagée

- Création d'un nouveau segment ou recherche de l'identifiant d'un segment déjà existant

```
#include<sys/ipc.h>
#include<sys/shm.h>
   int shmid = shmget(key_t cle, int taille, int flags);
```

  - retourne un entier positif (identificateur de segment dans la table) en cas de succès et -1 sinon.
  - Création si `cle = IPC_PRIVATE`
    `IPC_CREAT → flags`
    `IPC_CREAT | IPC_EXCL → flags`

Structure Associée

- La structure shmid_ds

```
struct  shmid_ds {
    struct      ipc_perm    shm_perm;      /* operation permission struct */
    uint   shm_segsz;                      /* size of segment in bytes */
    ushort shm_lpid;                       /* pid of last shmop */
    ushort shm_cpid;                       /* pid of creator */
    ushort shm_nattch;                     /* number of current attaches */
    time_t shm_atime;                      /* last  shmat  time */
    time_t shm_dtime;                      /* last  shmdt  time */
    time_t shm_ctime;                      /* last  change  time */
};
```

Initialisations Associées à une Création

- Création d'une nouvelle entrée
  - Initialisations

    `shm_perm.cuid` et `shm_perm.uid` ← `uid` effectif du processus appelant

    `shm_perm.cgid` et `shm_perm.gid` ← `gid` effectif du processus appelant

    `shm_perm.mode` ← 9 bits de poids faible de l'entier `flags`

    `shm_segsz` ← `taille`

    `shm_lpid`, `shm_nattch`, `shm_atime` et `shm_dtime` ← `0`

    `shm_ctime` ← heure courante

  - Création effective au premier attachement

Attachement et Détachement d'un Segment de Mémoire Partagée

- Attachement d'un segment

```
char *shmat(int shmid, char *adr, int flags);
```

  - rend l'adresse à laquelle le segment a été attaché
  - si premier attachement, alors allocation effective de l'espace mémoire correspondant
  - si `adr = 0` → le système choisit l'adresse d'attachement
  - possibilité d'attacher plus d'une fois un même segment par un processus
  - `SHM_RDONLY` → `flags` : `SIGSEGV` en cas de tentative d'écriture

- Détachement d'un segment

```
int shmdt(char *virtadr);
```

  - spécifier l'adresse et non pas l'identifiant
  - rend 0 en cas de succès et -1 sinon

Autres Appels Système

Segments de mémoire partagée attachés à un processus après un appel à :

- `fork()`
  - Héritage des segments de mémoire partagée par le fils
- `exec()`
  - Tous les segments de mémoire partagée sont détachés (pas détruits)
- `exit()`
  - Tous les segments de mémoire partagée sont détachés (pas détruits)

Opérations de Contrôle sur les Segments de Mémoire Partagée

- Opérations de contrôle avec la primitive `shmctl()`

```
#include<sys/shm.h>
    int shmctl(int shmid, int cmd, shmid_ds *buf);
```

cmd → IPC_STAT

→ IPC_SET
- shm_perm.uid
- shm_perm.gid
- shm_perm.mode

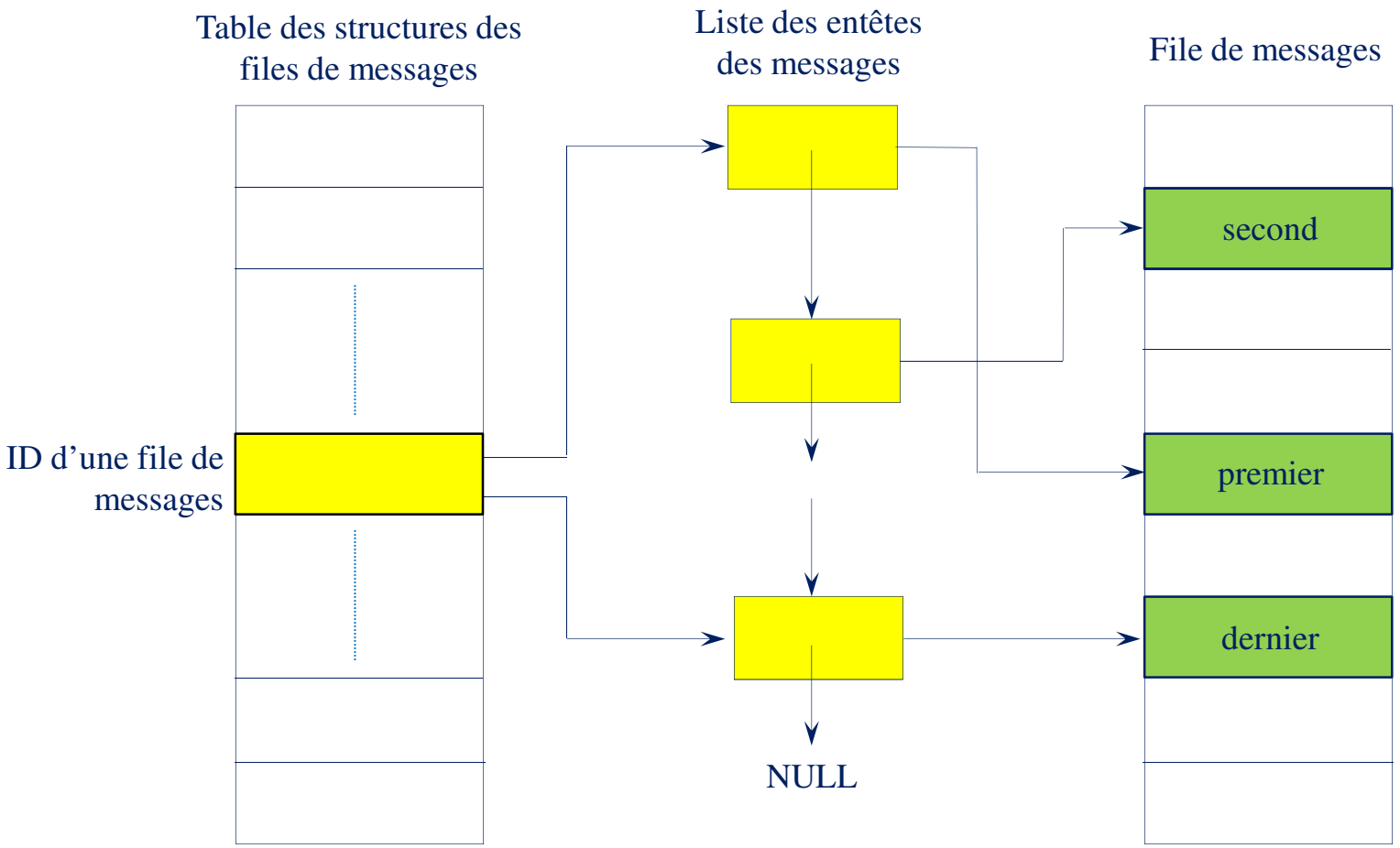
→ IPC_RMID

Opérations permises uniquement si `uid` effectif = super utilisateur
`shm_perm.cuid`
`shm_perm.uid`

Exemple

```
#include <sys/ipc.h>
#include <sys/shm.h>
…
int shm_id; struct shmid_ds *buf; key_t cle;
int *p_int; char *adr_att;
int taille = 1024;
…
cle = ftok("mem_par", 'M');
shm_id = shmget(cle, taille, 0666 | IPC_CREAT);
adr_att = shmat(shm_id, 0, 0600);
…
p_int = (int *)adr_att;
for (i=0; i<128; i++) *p_int++ = i;
…
shmdt(adr_att);
shmctl(shm_id, IPC_RMID, buf);
…
```

Les Files de Messages



Structure de données des messages

Caractéristiques des Files de Messages

- Emission et réception de flots de données entre processus
- Identifiées par des entiers
- Spécification de l'identifiant, et non pas le processus, lors d'une émission/réception
- Politique FIFO
- Un message = `Type` (entier dont l'interprétation est laissée à l 'utilisateur)
  + `Donnée` (chaîne de caractères de longueur quelconque)
- Connaissance + droits d'accès → droits d'émission/réception

Création d'une File de Messages

- Création d'une nouvelle file de messages ou recherche de l'identifiant d'une file déjà existante

```
#include<sys/ipc.h>
#include<sys/msg.h>
   int msgid = msgget(key_t cle, int flags);
```

  - retourne un entier positif (identifiant de la file de messages dans la table) en cas de succès et -1 sinon.
  - Création si `cle = IPC_PRIVATE`
    `IPC_CREAT → flags`
    `IPC_CREAT | IPC_EXCL → flags`

Structure Associée

- La structure msqid_ds

```
struct  msqid_ds {
    struct       ipc_perm   msg_perm;      /* operation permission struct */
    struct       msg    *msg_first;       /* ptr to first message on q */
    struct       msg    *msg_last;        /* ptr to last message on q */
    ushort msg_cbytes;                    /* current # of bytes on q */
    ushort msg_qnum;                      /* # of messages on q */
    ushort msg_qbytes;                    /* max # of bytes on q */
    ushort msg_lspid;                     /* pid of last  msgsnd */
    ushort msg_lrpid;                     /* pid of last  msgrcv */
    time_t msg_stime;                     /* last  msgsnd  time */
    time_t msg_rtime;                     /* last  msgrcv  time */
    time_t msg_ctime;                     /* last change time */
};
```

Initialisations Associées à une Création

- Initialisations lors de la création d'une nouvelle entrée
  - `msg_perm.cuid` et `msg_perm.uid` ← `uid` effectif du processus appelant
  - `msg_perm.cgid` et `msg_perm.gid` ← `gid` effectif du processus appelant
  - `msg_perm.mode` ← 9 bits de poids faible de l'entier `flags`
  - `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime` et `msg_rtime` ← `0`
  - `msg_qbytes` ← taille maximale permise par le système
  - `msg_ctime` ← heure courante

Émission d'un Message

- Primitive d'émission

```
#include<sys/msg.h>
    int msgsng(int msgid, struct msgbuf *msg, int taille,
                                                    int flags);
```

  - ```
    struct msgbuf {                 /* définie dans msg.h */
            long mtype;             /* type du message */
            char mtext[1];          /* texte du message */
    };
    ```

    Possibilité de redéfinir la structure `msgbuf` en fonction de ses besoins.

  - bloquante par défaut,
  - `mtype` ≤ 0, interdit en émission,
  - retourne 0 en cas de succès et -1 sinon.
    ```

Propriétés d'une Émission

- Émission bloquante (défaut)
  - Si file pleine, le processus est suspendu jusqu'à :
    - extraction de messages de la file,
    - suppression du système de la file (retourne `-1` et `errno` = `EIDRM`),
    - réception d'un signal.
  - Sinon,
    - insertion du message et de son type dans la file,
    - incrémentation du nombre de messages de la file,
    - mise à jour de l'identifiant du dernier écrivain,
    - mise à jour de la date de dernière écriture.

- Émission non bloquante
  - Si file pleine et `IPC_NOWAIT` → `flags`,
    - le message n'est pas envoyé et
    - le processus reprend immédiatement la main.

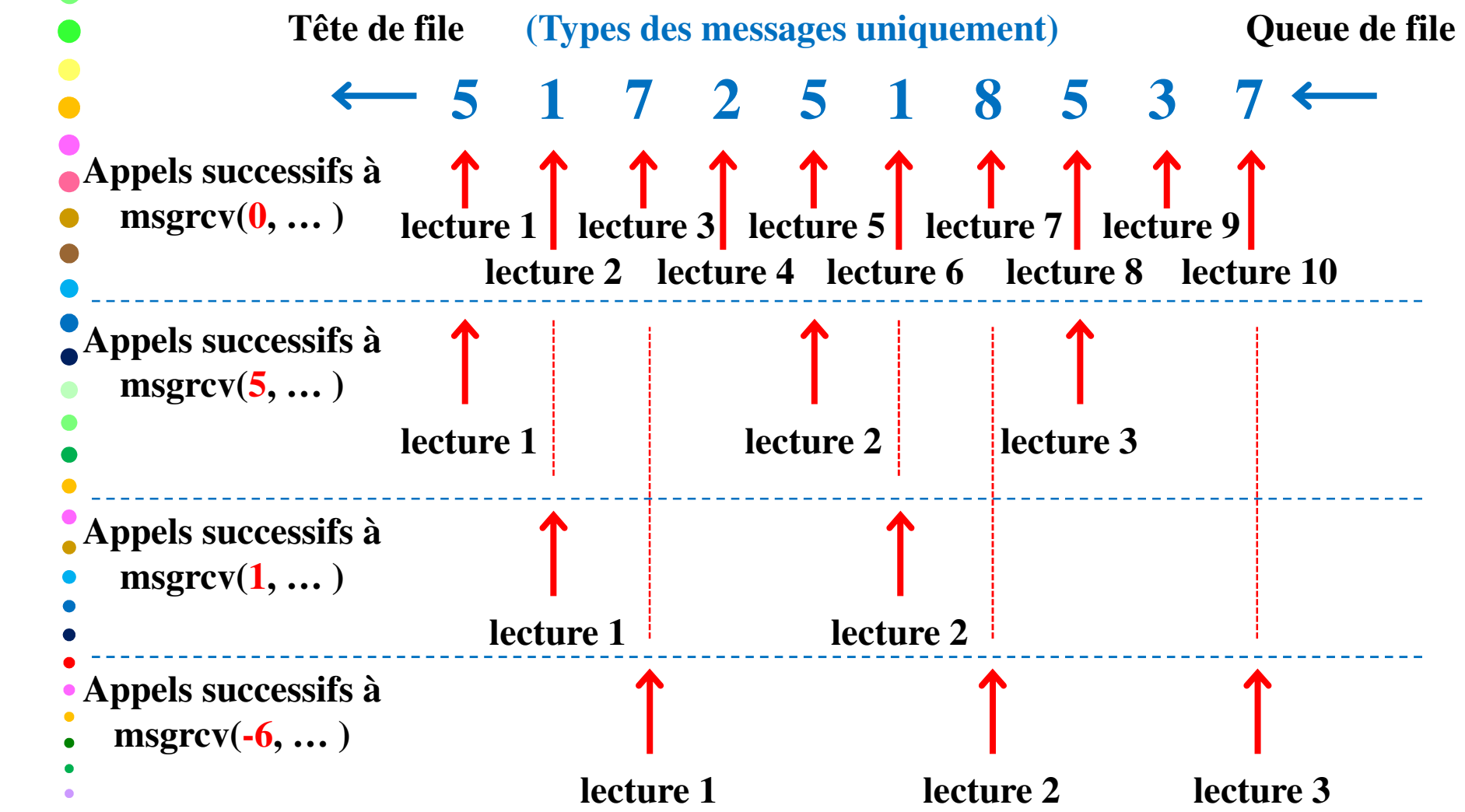
Extraction d'un Message d'une File

- Primitive de réception

```
#include<sys/msg.h>
    int msgrcv(int msgid, struct msgbuf *msg, int taille,
                                        long type, int flags);
```

  - Extraction quelconque ou sélective,
    `type` = `0` → le premier message de la file est extrait quel que soit son type,
    `type` > `0` → le premier message du type désigné est extrait,
    `type` < `0` → le premier message dont le type est supérieur à la valeur absolue du type désigné est extrait
  - bloquante par défaut,
  - si `taille` < taille du message, le système retourne une erreur (`errno` = `E2BIG`) et le message reste dans la file. Si `MSG_NOERROR` → `flags`, le système tronque le message sans générer d'erreur mais le reste du texte est perdu.
  - en cas de succès retourne le nombre de caractères dont est composé le texte du message et -1 sinon.

Exemples d'Extractions de Messages



Propriétés d'une Extraction

- Si aucun message ne répond aux conditions demandées :
  - `IPC_NOWAIT` $\not\subset$ `flags`, alors le processus est suspendu jusqu'à :
    - arrivée d'un message satisfaisant les conditions demandées,
    - suppression du système de la file (retourne -1 et errno = EIDRM),
    - réception d'un signal.
  - `IPC_NOWAIT` $\subset$ `flags`, alors :
    - le processus reprend immédiatement la main,
    - retourne -1 et errno = ENOMSG.

- Sinon,
  - extraction effective du message de la file,
  - décrémentation du nombre de messages de la file,
  - mise à jour de l'identifiant du dernier lecteur,
  - mise à jour de la date de dernière lecture.

Contrôle de l'État d'une File

- Opérations de contrôle avec la primitive `msgctl()`

```
#include<sys/msg.h>
    int msgctl(int msgid, int cmd, msqid_ds *buf);
```

– consultation, modification des caractéristiques et suppression d'une file

`cmd` → `IPC_STAT`

→ `IPC_SET`
- `msg_perm.uid`
- `msg_perm.gid`
- `msg_perm.mode`
- `msg_qbytes`

→ `IPC_RMID`

Opérations permises uniquement si `uid` effectif = super utilisateur
`shm_perm.cuid`
`shm_perm.uid`

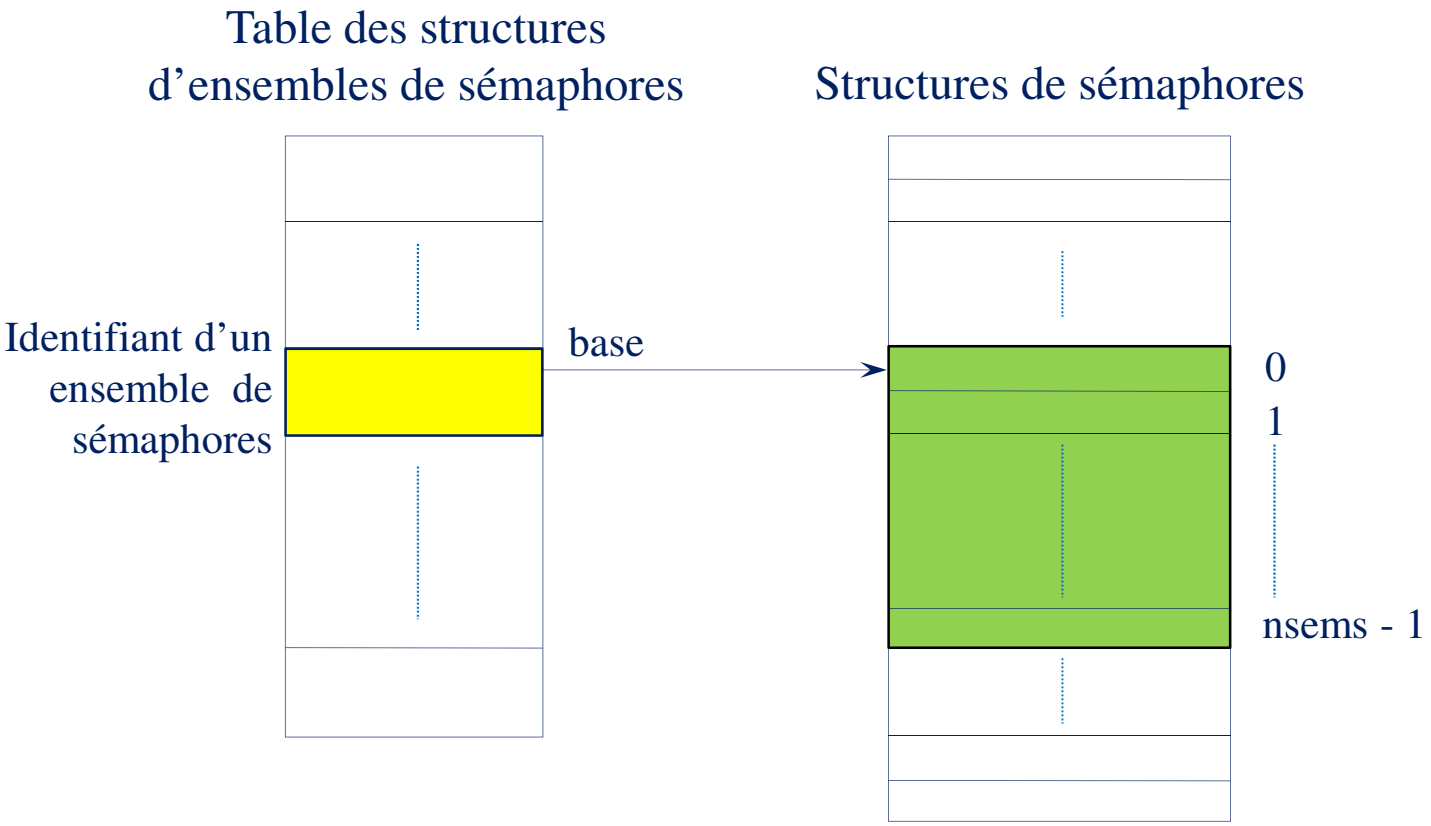
Modification `msg_qbytes` → root

– retourne 0 en cas de succès et -1 sinon

Exemple

```
#include <sys/ipc.h>
#include <sys/msg.h>
…
int msg_id; struct msqid_ds *buf; key_t cle;
struct message {   long type;
                   char texte[128]; } msg;
char path[14]= "file_msg"; char code='Q';
…
cle = ftok(path, code);
msg_id = msgget (cle, 0666 | IPC_CREAT);
…
msg.type = 1;
for ( ; ; ) {
   printf ( "Entrer le texte a emettre \n");
   scanf("%s",msg.texte);
   msgsnd(msg_id , &msg , strlen( msg.texte ) , 0);
   …
}
…
```

Les Sémaphores



Structure de données des sémaphores

Problème

Compte Bancaire : A

```
Crediter (X) {

1. tmp ← A
2. tmp ← tmp + X
3. A ← tmp

}
```

Section Critique (SC)

```
Debiter (Y) {

4. tmp ← A
5. tmp ← tmp - Y
6. A ← tmp

}
```

Exécutions séquentielles:

Crediter(X); Debiter(Y); ⇒ A + X - Y

Debiter(Y); Crediter(X); ⇒ A - Y + X

} Corrects

Exécution : 1. 2. 4. 5. 6. 3. Résultat = A + X

Exécution : 4. 1. 2. 3. 5. 6. Résultat = A - Y

Les sémaphores

Ressource partagée : R

- pour gérer les accès concurrentiels à la ressource R, on lui associe un sémaphore
- un sémaphore (Edsger Dijkstra en 1965) c'est :
  - un compteur,
  - une file de processus en attente.
- le compteur n'est manipulé que par :
  - init(),
  - P() et V().

Réserver la ressource

```
P() {
    si ( compteur > 0 ){
       compteur = compteur - 1
    } sinon {
       endormir le processus
    }
}
```

Libérer la ressource

```
V() {
    compteur = compteur + 1
    si (file non vide ){
       réveillez processus
    }
}
```

La solution

```
Compte Bancaire : A  ← sémaphore avec compteur = 1
```

```
P() {
si ( compteur > 0 ){
   compteur = compteur - 1
} sinon {
   endormir le processus
}
}
```

```
Crediter (X) {
P()
tmp ← A
tmp ← tmp + X
A ← tmp
V()
}
```

```
Debiter (Y) {
P()
tmp ← A
tmp ← tmp - Y
A ← tmp
V()
}
```

Problème repoussé mais inchangé

Résolution du problème

- **Atomicité des primitives P() et V()**
- **P() et V() s'exécutent en totalité sans pouvoir être interrompues, ou pas du tout**

Généralisation

Si compteur sémaphore initialement = 1
- sémaphore binaire = mutex
- exclusion mutuelle

Si compteur sémaphore initialement = N
- sémaphore de synchronisation
- gestion d'exemplaires d'une ressource, par exemple

Généralisation :
- sémaphore sem
  - de compteur sem.compteur et
  - de file sem.file

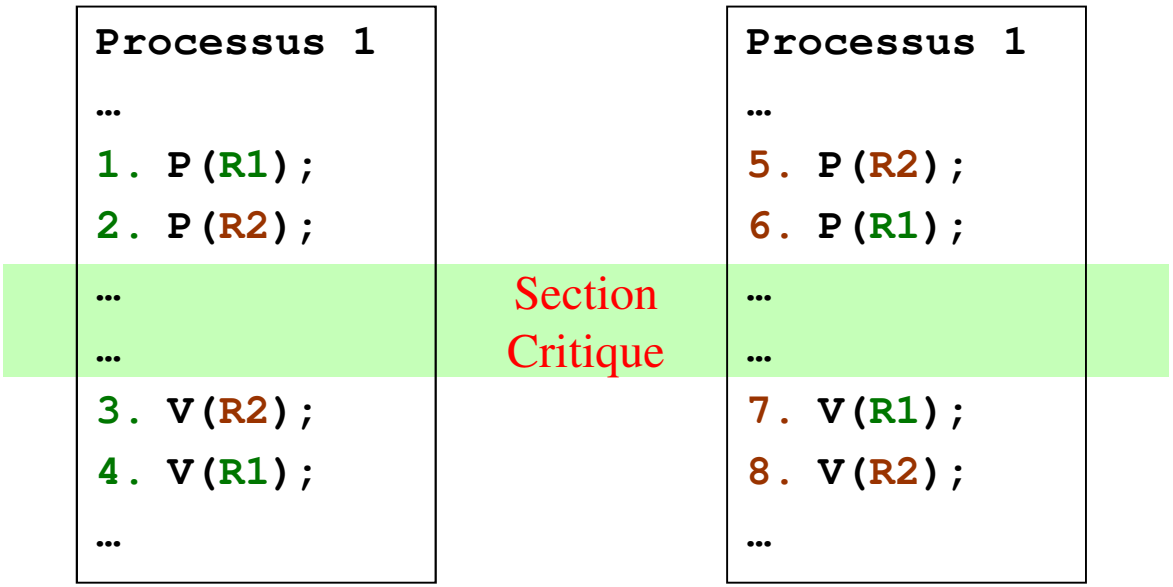
```
P(sem, X) {
si ( sem.compteur >= X ){
        sem.compteur = sem.compteur - X
    } sinon {
        endormir le processus
    }
}
```

```
V(sem, Y) {
    sem.compteur = sem.compteur + Y
    si (sem.file non vide ){
        réveillez processus
    }
}
```

Problème d'interblocage

Supposons que 2 processus, pour réaliser leur tâche, nécessitent l'accès exclusif à 2 ressources R1 et R2.

Si l'allocation des 2 ressources est inversée chez les 2 processus :



Scénario : 1. 5. 2. 6. ⇨ Interblocage
Solution : cf implémentation

Caractéristiques des Ensembles de Sémaphores

- Mécanisme de synchronisation
  - accès concurrents à une ressource partagée
  - solution au problème de l'exclusion mutuelle
- Un sémaphore
  - → un compteur
  - → une file d'attente
- Acquisition simultanée d'exemplaires multiples de plusieurs ressources différentes
- Identifiés par des entiers

Création d'un Ensemble de Sémaphores

- Création d'un nouvel ensemble de sémaphores ou recherche de l'identifiant d'un ensemble de sémaphores

```
#include<sys/ipc.h>
#include<sys/sem.h>
    int sem_id = semget(key_t cle, int nsems, int flags);
```

  - retourne un entier positif (identifiant de l'ensemble des sémaphores dans la table) en cas de succès et -1 sinon.
  - création si `cle = IPC_PRIVATE`
    `IPC_CREAT → flags`
    `IPC_CREAT | IPC_EXCL → flags`
  - Création d'un *ensemble* de sémaphores qui auront tous les mêmes droits

Structures Associées

- Ensemble de sémaphores

```
struct semid_ds {
    struct      ipc_perm    sem_perm;   /* operation permission struct */
    struct      sem   *sem_base;        /* ptr to first semaphore in set */
    ushort      sem_nsems;              /* # of semaphores in set */
    time_t      sem_otime;              /* last semop time */
    time_t      sem_ctime;              /* last change time */
};
```

- Sémaphore individuel

```
struct sem {
    ushort      semval;                 /* semaphore text map address */
    short       sempid;                 /* pid of last operation */
    ushort      semncnt;                /* # awaiting semval > cval */
    ushort      semzcnt;                /* # awaiting semval = 0 */
};
```

Initialisations Associées à une Création

- Initialisations lors de la création d'une nouvelle entrée
  - `sem_perm.cuid` et `sem_perm.uid` ← `uid` effectif du processus appelant
  - `sem_perm.cgid` et `sem_perm.gid` ← `gid` effectif du processus appelant
  - `sem_perm.mode` ← 9 bits de poids faible de l'entier `flags`
  - `sem_nsems` ← `nsems`
  - `sem_otime` ← `0`
  - `sem_ctime` ← heure courante

Opérations sur les Sémaphores

- La primitive `semop()`

```
#include<sys/ipc.h>
#include<sys/sem.h>
    int semop(int semid, struct sembuf *sops, unsigned nsops);

struct sembuf {
    ushort sem_num;          /* semaphore # */
    short  sem_op;           /* semaphore operation */
    shrot  sem_flag;         /* operation flags */
};
```

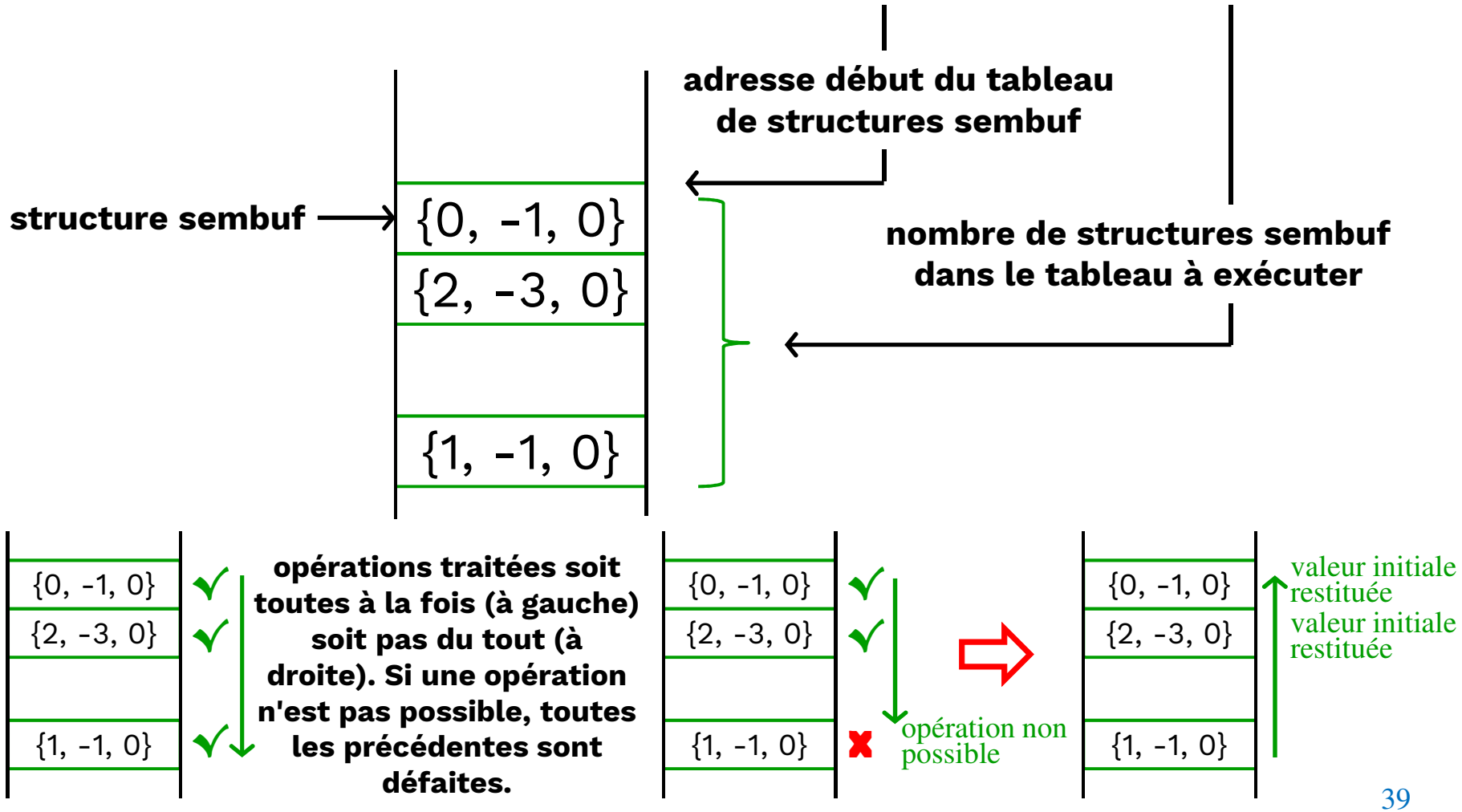
  - chaque opération précisée par `sem_op` est exécutée sur le sémaphore correspondant spécifié par `semid` et `sem_num`,
  - opérations traitées soit **toutes** à la fois soit pas du tout,
  - si un processus est obligé de s'endormir, la valeur initiale des sémaphores (avant l'appel) est restituée,
  - retourne 0 en cas de succès et -1 sinon.

Nature des Opérations sur les Sémaphores

sem_op < 0	semval ≥ | sem_op |	semval := semval - | sem_op | sem_flg & SEM_UNDO est “vrai” semadj := semadj + | sem_op |
	semval < | sem_op |	sem_flg & IPC_NOWAIT est “faux” semncnt := semncnt + 1 état (process) = suspendu
		sem_flg & IPC_NOWAIT est “vrai” Retour immédiat
sem_op > 0	semval := semval + sem_op sem_flg & SEM_UNDO est “vrai” semadj := semadj - sem_op	
sem_op = 0	semval = 0	Retour immédiat
	semval ≠ 0	sem_flg & IPC_NOWAIT est “faux” semzcnt := semzcnt + 1 état (process) = suspendu
		sem_flg & IPC_NOWAIT est “vrai” Retour immédiat

Opérations sur les Sémaphores

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



Structure undo

- La structure sem_undo

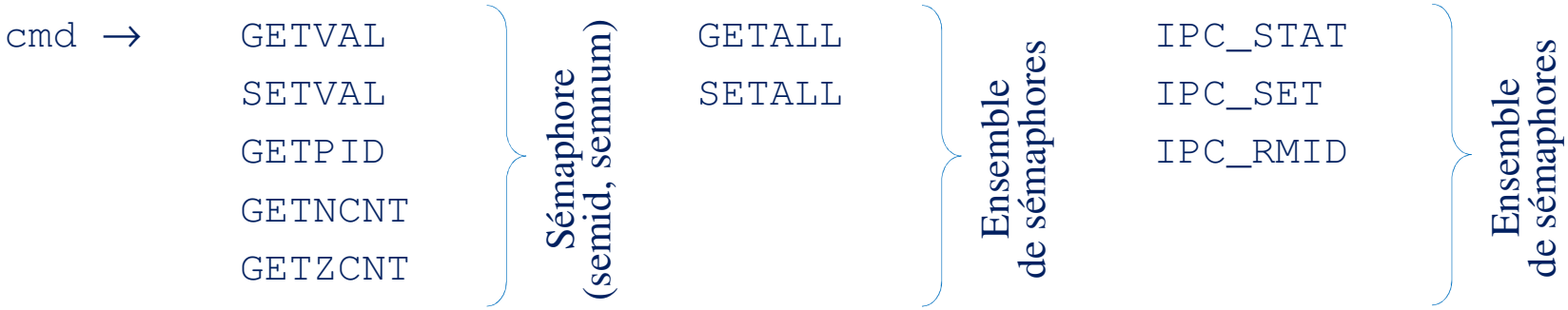
```
struct sem_undo {
    struct sem_undo *un_np;     /* ptr to next active undo structure */
    short  un_cnt;              /* # of active entries */
    struct undo {
          short un_aoe;         /* adjust on exit values */
          short un_num;         /* semaphore # */
          int   un_id;          /* semid */
    }     un_ent[1];            /* (semume) undo entries (one minimum) */
};
```

Opérations de Contrôle sur les Sémaphores

- Les opérations de contrôle avec la primitive `semctl()`

```
#include<sys/ipc.h>
#include<sys/sem.h>
    int semctl(int semid, int semnum, int cmd, union semun arg);

union semun {
    int    val;
    struct semid_ds *buf;
    u_short *array;
};
```



Exemple

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

#define      SEM_EXCL_MUT       0
#define      NB_SEM             1
char sem_path[14] = "ens_sem"; char sem_code = 'S';
int FLAGS = 0666 | IPC_CREAT; key_t   sem_cle; int  sem_id;
struct  sembuf  operation;

main ( ) {
sem_cle = ftok(sem_path, sem_code);
sem_id = semget (sem_cle , NB_SEM, FLAGS );
semctl(sem_id, SEM_EXCL_MUT, SETVAL, 1);
…
   P(SEM_EXCL_MUT);
       /* Section Critique */
   V(SEM_EXCL_MUT);
…
```

Exemple (suite)

```
void P (sem)                          /* Primitive P() sur sémaphores */
   int  sem; {                        /* Identifiant du sémaphore */
   operation.sem_num = sem;           /* Identification du sémaphore impliqué  */
   operation.sem_op = -1;             /* Définition de l'opération à réaliser  */
   operation.sem_flg = SEM_UNDO;      /* Positionnement du bit SEM_UNDO */
   semop (sem_id, &operation, 1);     /* Exécution de l'opération définie */
};

void V(sem)                           /* Primitive V() sur sémaphores */
   int  sem; {
   operation.sem_num = sem;
   operation.sem_op = 1;
   operation.sem_flg = SEM_UNDO;
   semop (sem_id, &operation, 1);
};
```