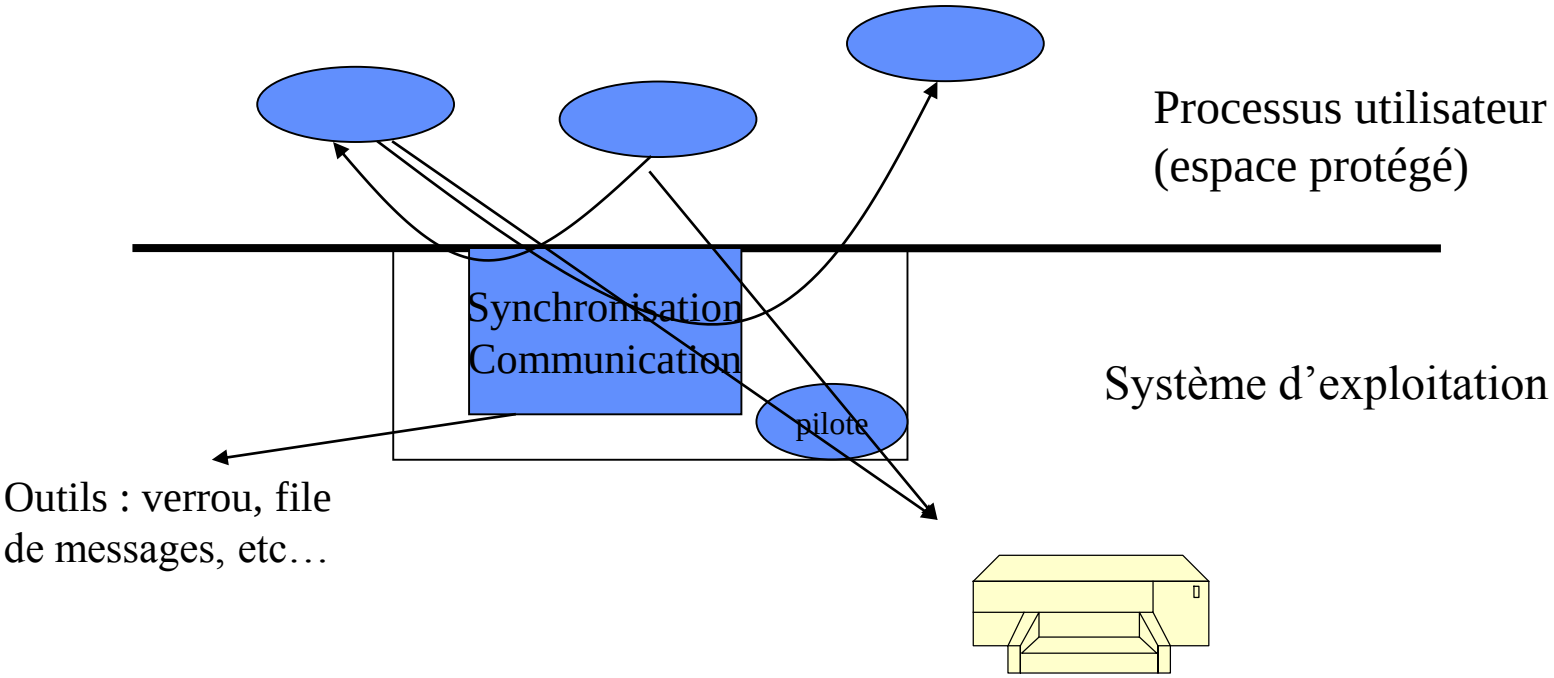
Synchronisation entre processus



INTRODUCTION

- **Nous nous intéressons au développement d'applications multiprocessus concurrents c'est à dire d'applications composées de plusieurs processus indépendants et en concurrence pour l'accès aux ressources du système.**

Les processus sont ordonnancés indépendamment les uns des autres.

Une ressource désigne toute entité dont a besoin un processus pour s'exécuter.

- Ressource matérielle (processeur, périphérique)
- Ressource logicielle (fichier)

Notion de ressources

- Définitions

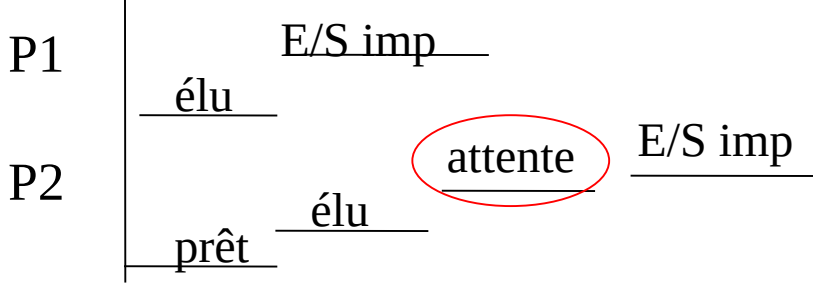
  – Une ressource désigne toute entité dont a besoin un processus pour s'exécuter.
    - Ressource matérielle (processeur, périphérique)
    - Ressource logicielle (fichier, variable).

  – Une ressource est caractérisée
    - par un état : libre / occupée
    - par son nombre de points d'accès (nombre de processus pouvant l'utiliser en même temps)

Notion de ressources

- Utilisation d'une ressource par un processus
  - Trois étapes : Allocation
    Utilisation
    Restitution
  - Les phases d'allocation et de restitution doivent assurer que le ressource est utilisée conformément à son nombre de points d'accès
    - ressource critique à un seul point d'accès

Notion de ressources
Exemple

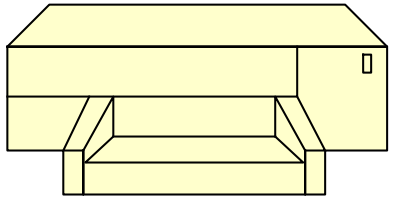


- Ressource matérielle : imprimante

Processus P1

Print (texte)

SE
Gestion des entrées/sorties
PRINT



LIBRE
1 point d'accès

Print (texte)

Allouée P1

OCCUPEE
1 point d'accès

Processus P2

P2 en attente jusqu'à
libération par P1

Outil de synchronisation : le verrou

- Un mécanisme proposé pour permettre de résoudre les concurrences d'accès à une ressource est le mécanisme de *verrou.* Un verrou est un objet système à deux états (libre/occupé) sur lequel deux opérations sont définies.

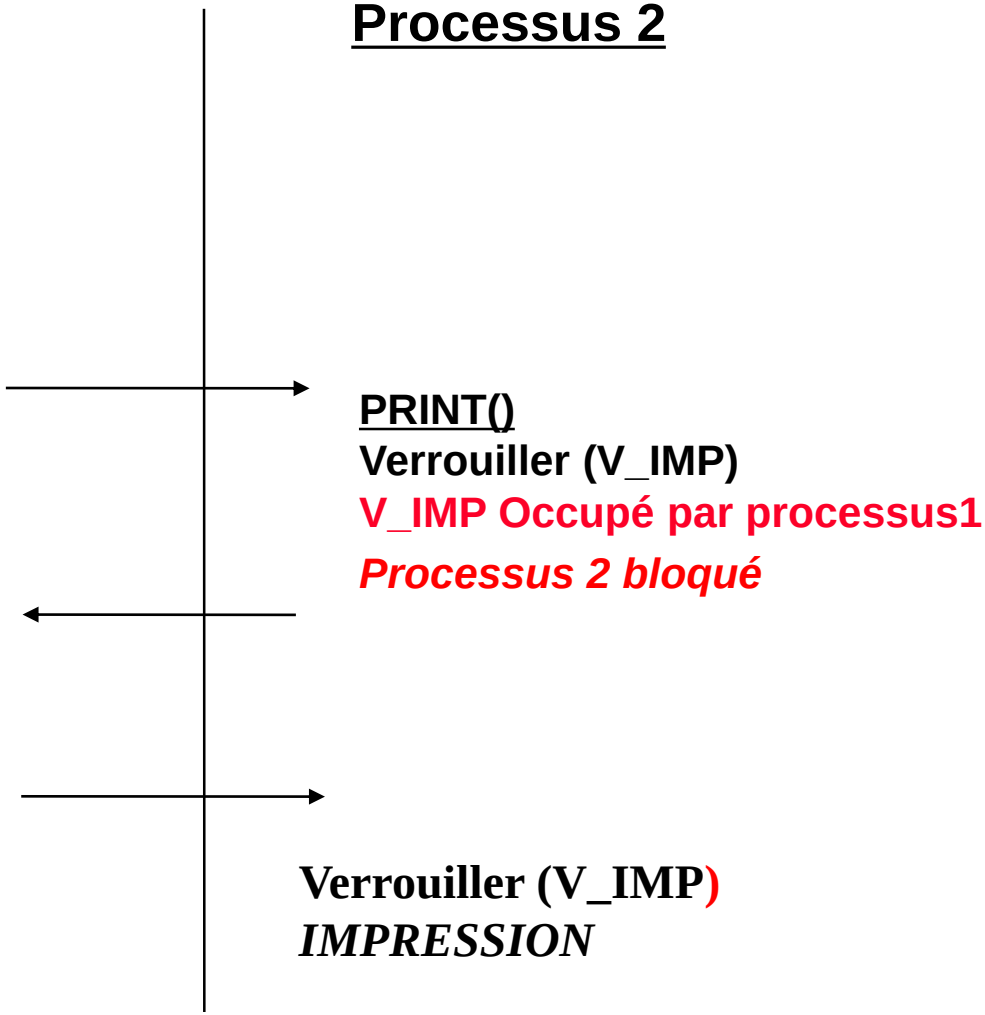
  – *verrouiller (v)* permet au processus d'acquérir le verrou *v* s'il est libre. S'il n'est pas disponible, le processus est bloqué en attente de la ressource.

  – *déverrouiller (v)* permet au processus de libérer le verrou *v* qu'il possédait. Si un ou plusieurs processus étaient en attente de ce verrou, un seul de ces processus est réactivé et reçoit le verrou.

- En tant qu'opérations systèmes, ces opérations sont indivisibles, c'est-à-dire que le système qu'elles s'exécutent interruptions maquées.

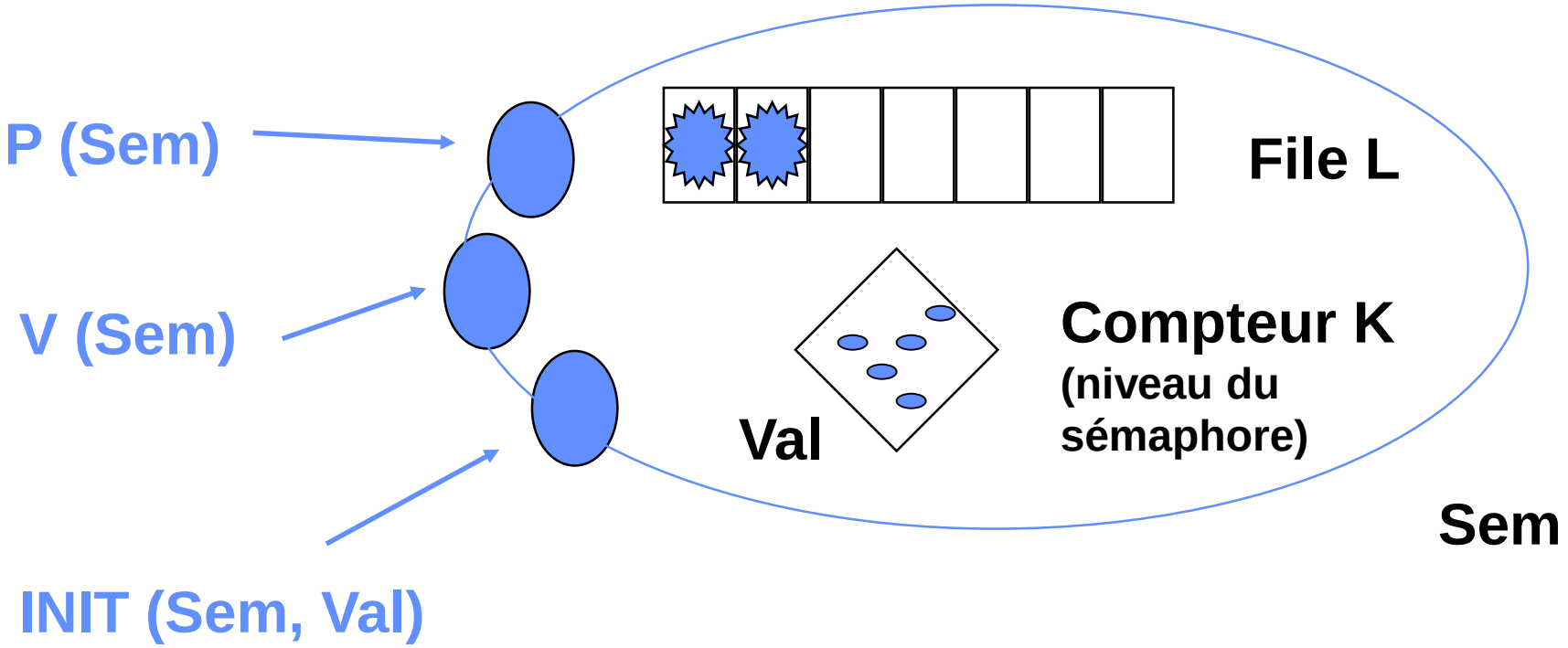
Outil de synchronisation : le verrou

V_IMP : verrou; -- verrou libre



Les sémaphores

- **Le système d'exploitation offre un outil appelé sémaphore, constitué d'un compteur K et d'une file L**
- **Trois opérations atomiques sont appliqués au semaphore Sem:**



Les sémaphores

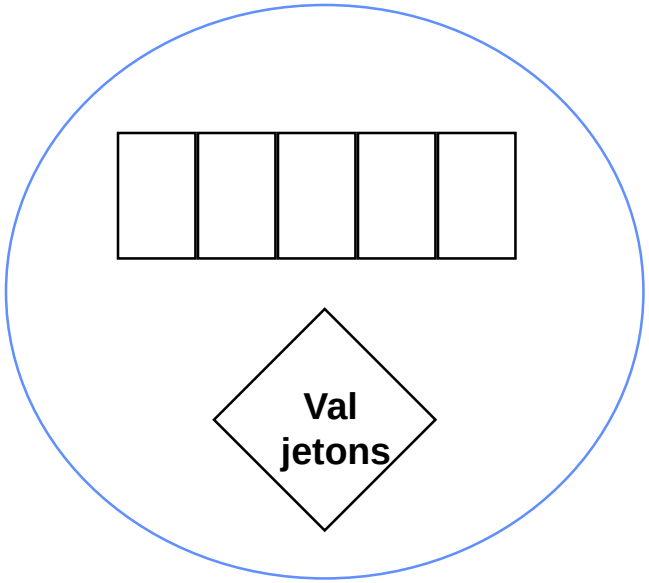
- **Un sémaphore Sem peut être vu comme un distributeur de jetons; le jeton étant un droit à poursuivre son exécution**
  - **l'opération INIT (Sem, Val) fixe le nombre de jetons initial**
  - **l'opération P(Sem) attribue un jeton au processus appelant si il en reste sinon bloque le processus dans Sem.L**
  - **l'opération V(Sem) restitue un jeton et débloque un processus de Sem.L si il en existe un.**

Les sémaphores

- **Opération Init (Sem, Val)**

Init (Sem, Val)

début
masquer_it
Sem. K := Val ***(jetons);***
Sem. L := ∅
demasquer_it
fin

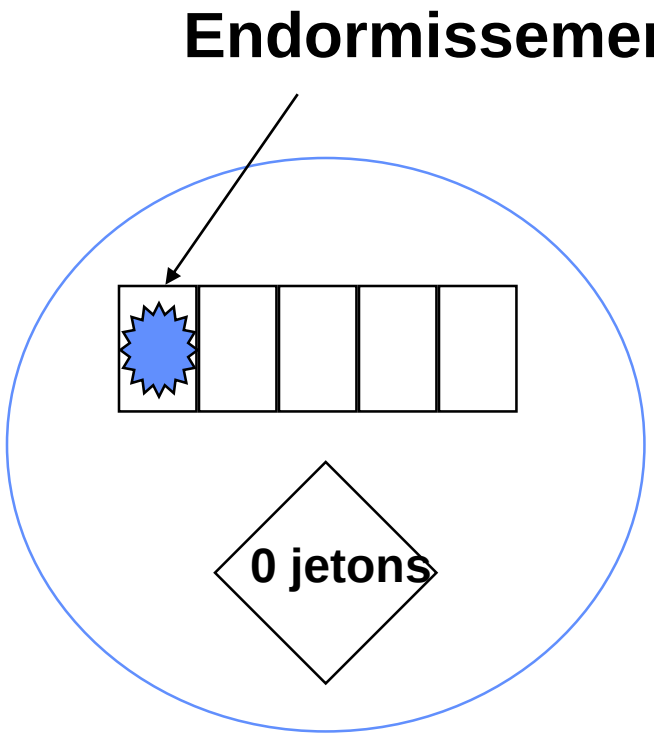


Les sémaphores

- **Opération P (Sem)**

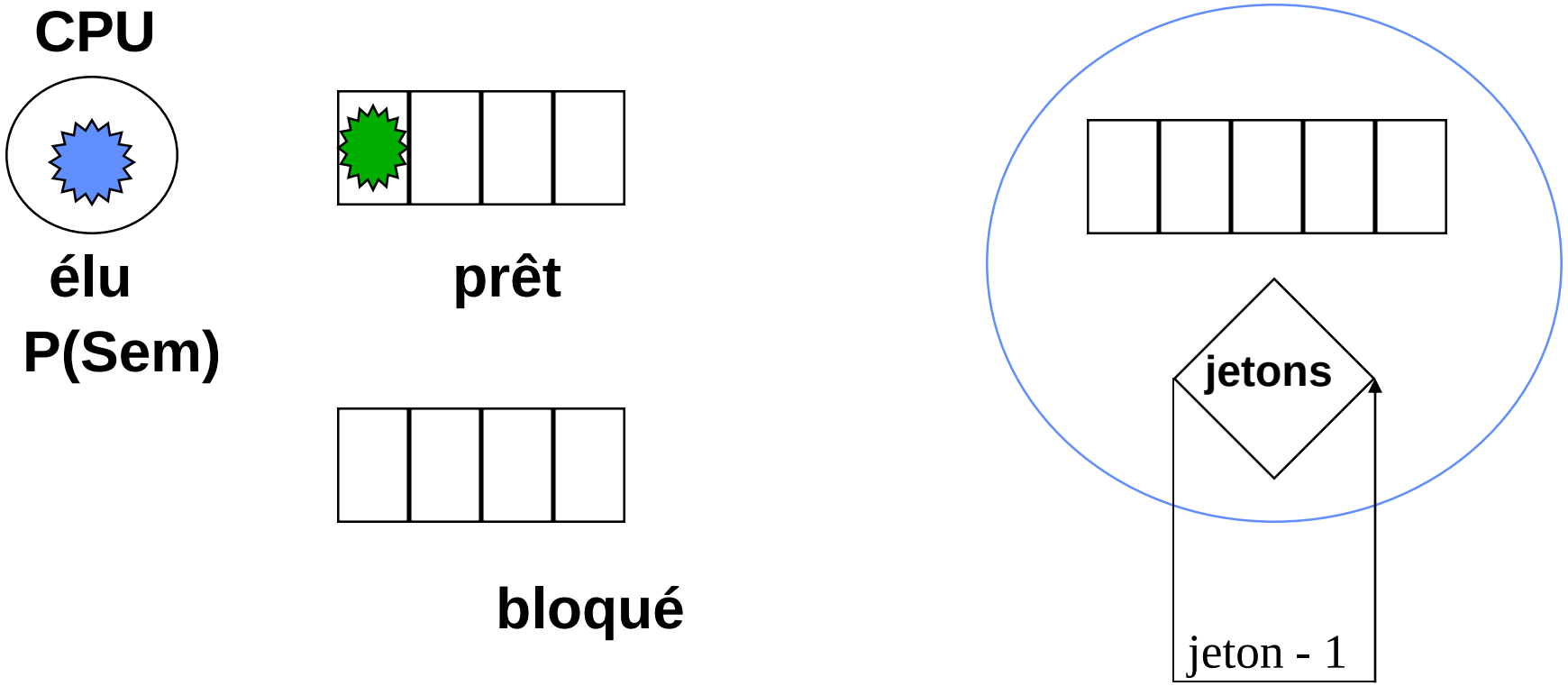
P (Sem)

début
masquer_it
Si (Sem.K > 0) alors
Sem.K = Sem.K – 1; **(donner jeton)**
sinon
ajouter ce processus à Sem.L
bloquer ce processus
fsi
demasquer_it
fin



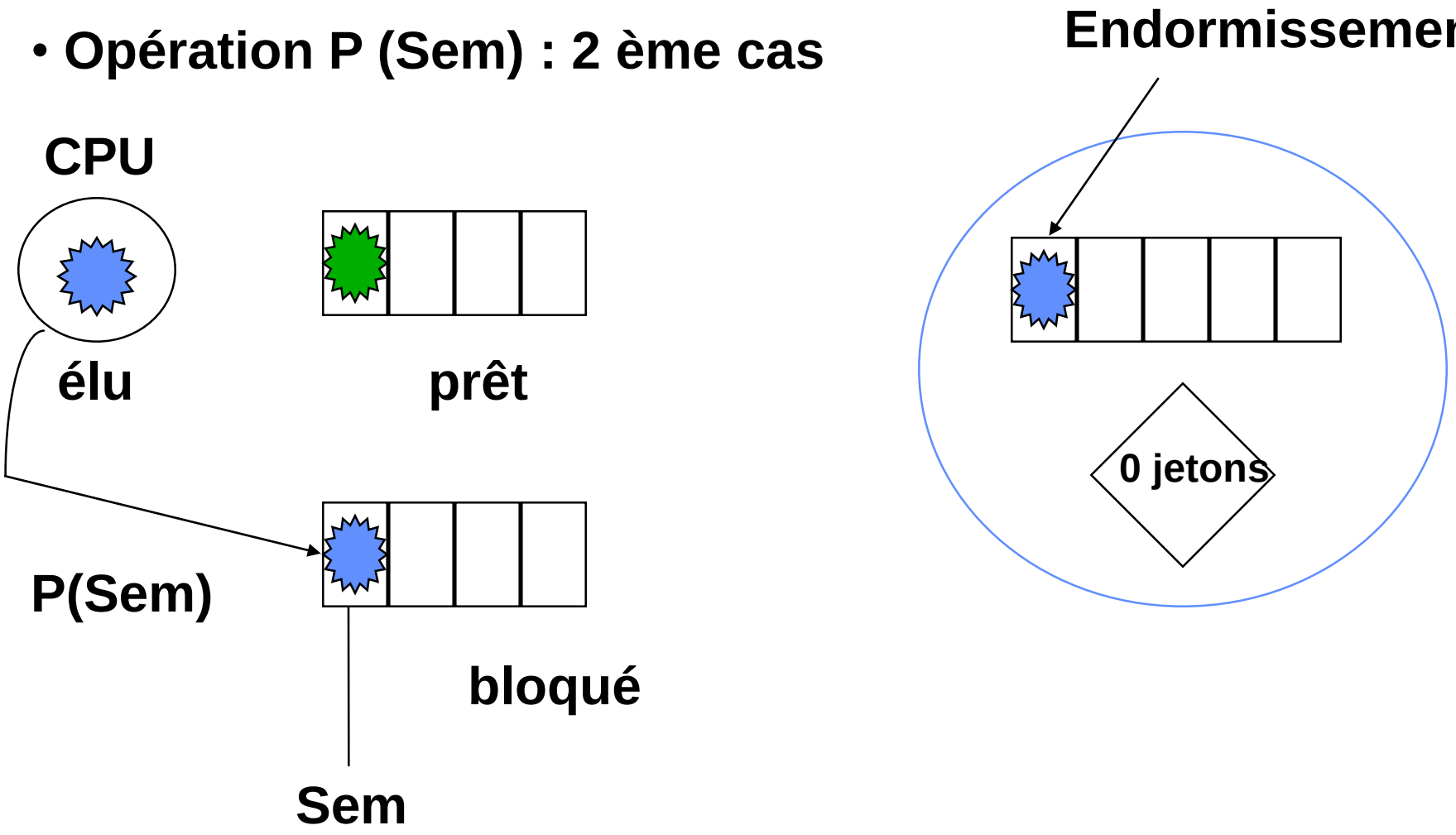
Les sémaphores

- **Opération P (Sem) : 1er cas**



Les sémaphores

- **Opération P (Sem) : 2 ème cas**

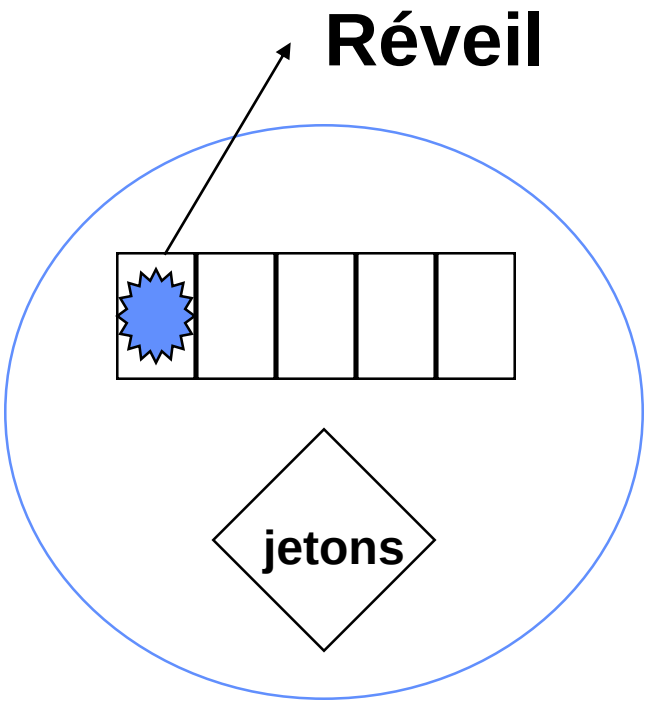


Les sémaphores

- **Opération V (Sem)**

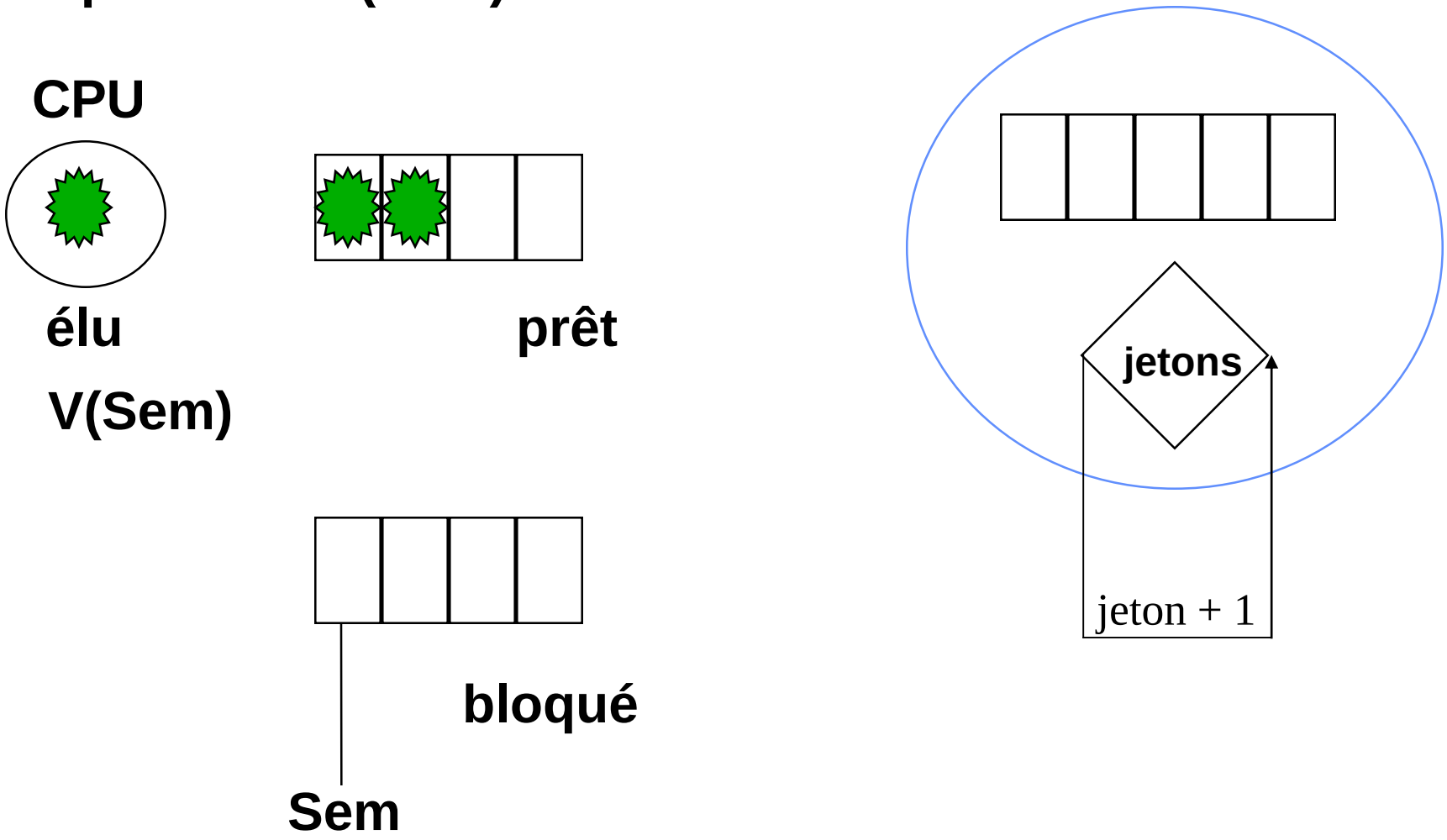
V (Sem)

début
masquer_it
Sem.K = Sem.K + 1; (rendre jeton)
Si il y a un processus en attente de jeton dans Sem.L
alors
sortir un processus de Sem.L
Sem.K = Sem.K – 1; *(donner jeton)*
réveiller ce processus
fsi
demasquer_it
fin



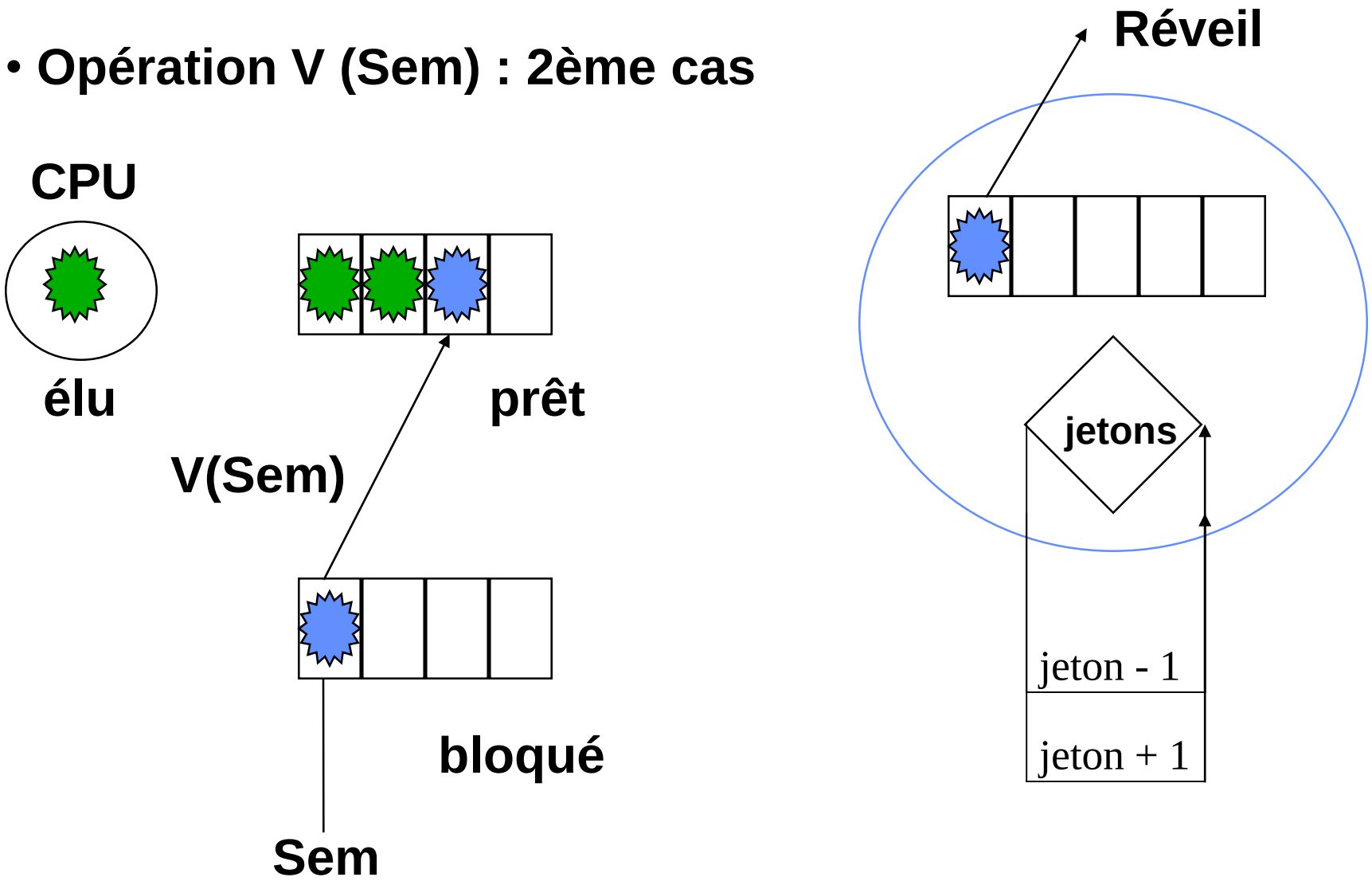
Les sémaphores

- **Opération V (Sem) : 1er cas**



Les sémaphores

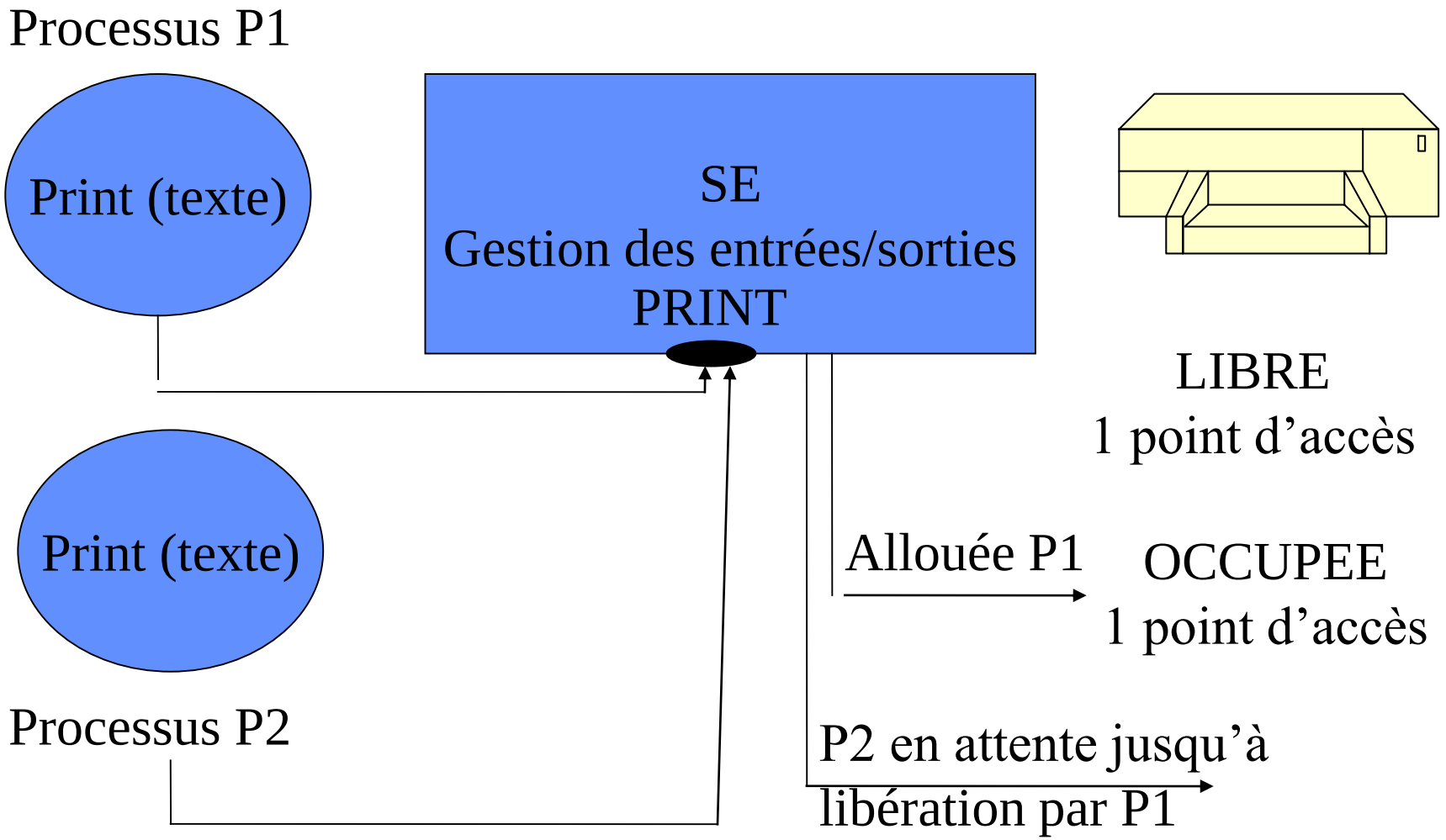
- **Opération V (Sem) : 2ème cas**



Notion de ressources
Exemple

ressource critique à un seul point d'accès

- Ressource matérielle : imprimante



Section critique et exclusion mutuelle

Processus
Début

Entrée Section Critique

Ressource Critique
IMPRIMANTE

Sortie Section Critique
Fin

- **Ressource utilisable par un seul processus à la fois**

SECTION CRITIQUE
(code d'utilisation de la ressource critique)

☞ L'entrée et la sortie de SC doivent assurer qu'à tout moment, un seul processus s'exécute en SC (exclusion mutuelle)

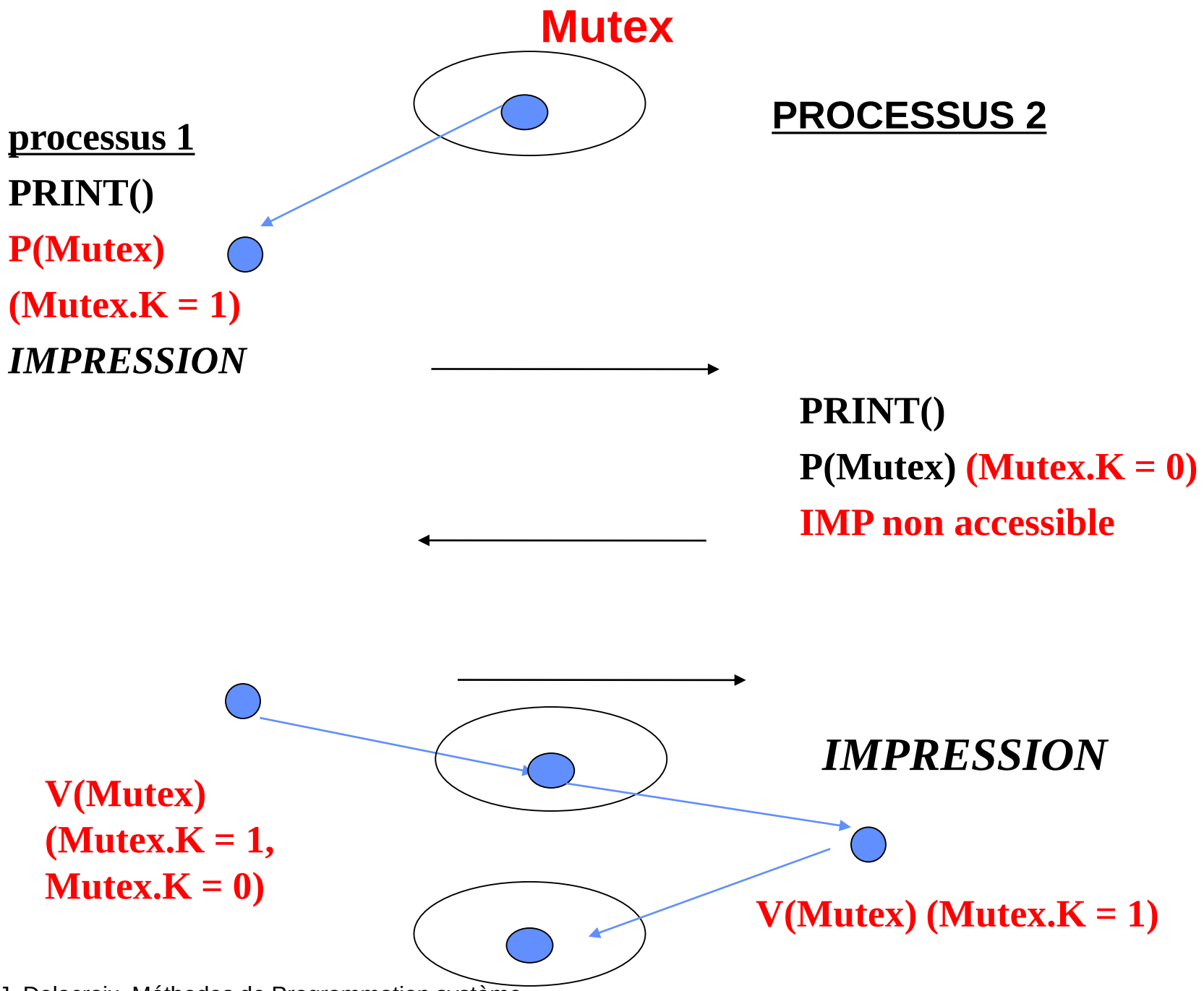
Section critique avec sémaphore

1 seul processus en section critique
=> 1 seul jeton
Sémaphore Mutex initialisé à 1

P (Mutex) — **Entrée section_critique**

Section Critique

V (Mutex) — **Sortie section_critique**

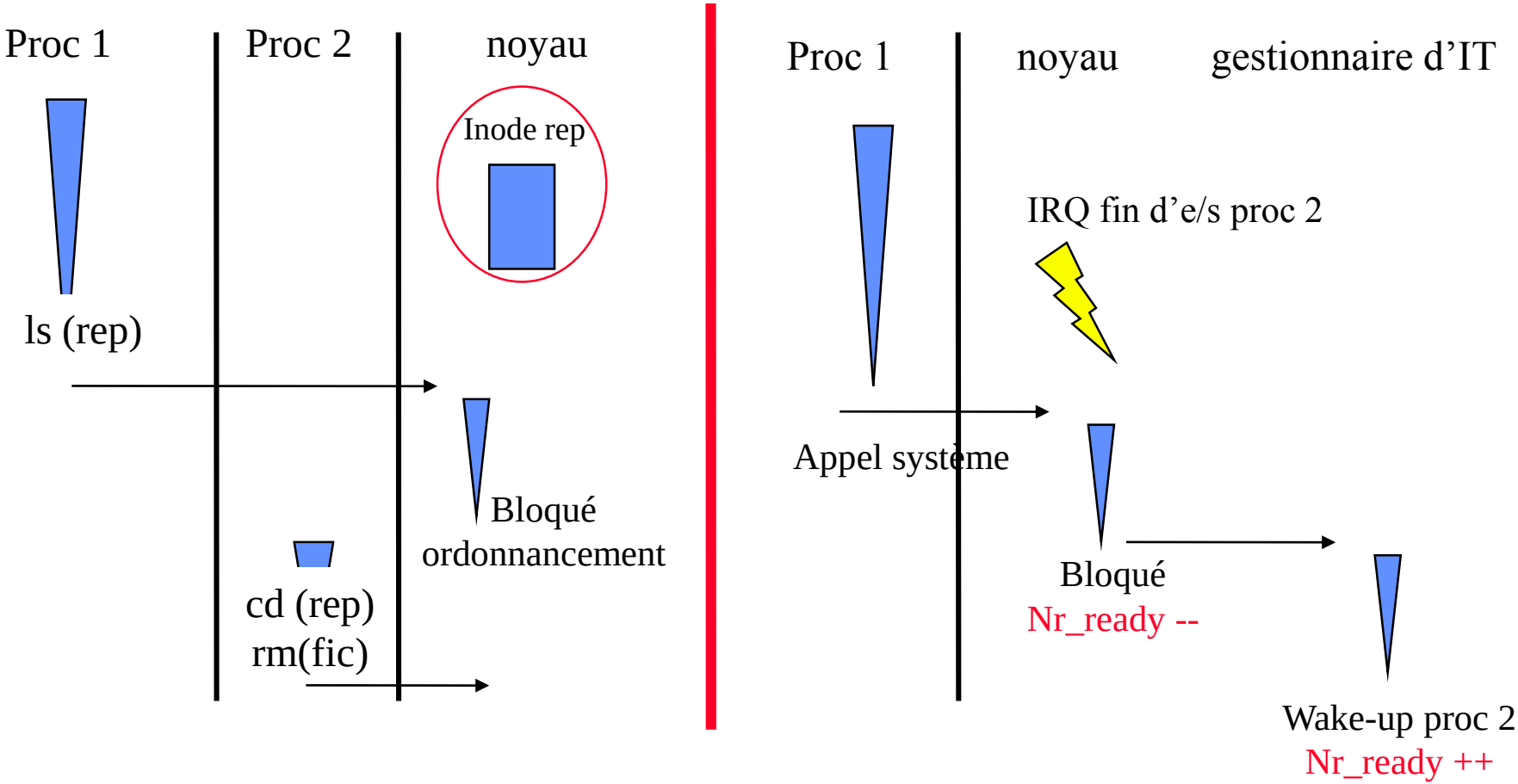

Mutex
processus 1
PRINT()
P(Mutex)
(Mutex.K = 1)
IMPRESSION
PROCESSUS 2
PRINT()
P(Mutex) (Mutex.K = 0)
IMP non accessible
IMPRESSION
V(Mutex)
(Mutex.K = 1,
Mutex.K = 0)
V(Mutex) (Mutex.K = 1)

Synchronisation et communication entre processus

Synchronisation au sein du noyau Linux

Synchronisation au sein du noyau Linux

- Les chemins de contrôle du noyau peuvent s'imbriquer et créer des situations de concurrence d'accès sur les structures de données du noyau.



Synchronisation au sein du noyau Linux

- Les chemins de contrôle du noyau peuvent s'imbriquer et créer des situations de concurrence d'accès sur les structures de données du noyau.

- Mise en œuvre de techniques de synchronisation au sein du noyau :

  – Opérations atomiques en assembleur

Nr_ready ++ compilateur

```
load D R1 Nr_ready
add Im R1 1
store D R1 Nr_ready
```

définition de macros utilisant des instructions atomiques assembleur

Atomic_int (Nr_ready)
{inc (Nr_ready)}

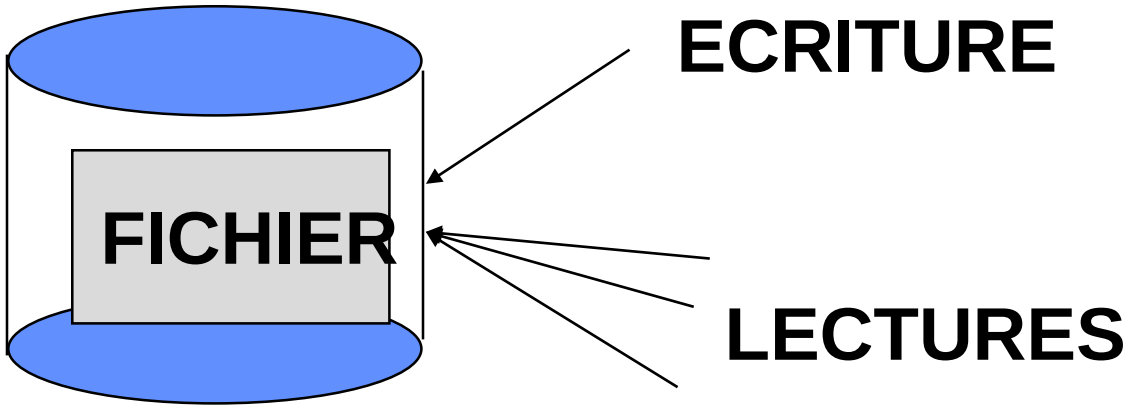
  – Masquage des interruptions (_cli, _sti)

  Notamment lors de la manipulation des files de l'ordonnanceur

  – Sémaphores du noyau

  Structure composée d 'un compteur et d'une file; opération down (P) et up (V), utilisée notamment pour protéger les descripteurs de régions et les inodes

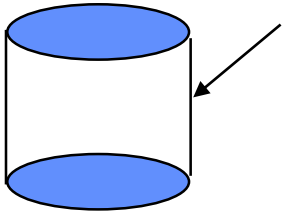
Les sémaphores
Lecteurs / Rédacteurs



- **Ecriture seule ou Lectures simultanées**

  ☞ **Un rédacteur exclut** **- les rédacteurs**
  - les lecteurs

  ☞ **Un lecteur exclut** **- les rédacteurs**



Les sémaphores Lecteurs / Rédacteurs

- **Un rédacteur exclut les rédacteurs et les lecteurs**

☞ **Un rédacteur effectue des accès en exclusion mutuelle des autres rédacteurs et des lecteurs**

➔ **Sémaphore d'exclusion mutuelle <u>Accès</u> initialisé à 1**

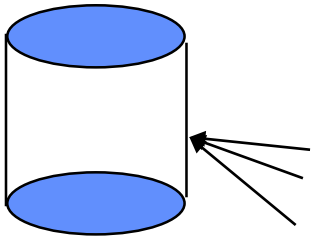
Les sémaphores
Lecteurs / Rédacteurs

Rédacteur

M'assurer que l'accès au fichier est libre
(pas de lecteurs, pas de rédacteur)
P(Accès)

entrer en écriture

Libérer l'accès au fichier (pour un rédacteur ou un lecteur)
V(Accès)



LECTURES

Les sémaphores Lecteurs / Rédacteurs

- **Un lecteur exclut les rédacteurs**

  ☞ **Un premier lecteur doit s'assurer qu'il n'y a pas d'accès en écriture en cours**

  ☞ **Le dernier lecteur doit réveiller un éventuel rédacteur**

➔ **NL, nombre de lecteurs courants, initialisé à 0**

Les sémaphores
Lecteurs / Rédacteurs

Lecteur

Compter un lecteur de plus
Si je suis le premier lecteur alors
Y-a-t-il un rédacteur ?
si oui, attendre
fsi

NL : = NL + 1

Si NL = 1 alors

P(Accès)
fsi

entrer en lecture

Compter un lecteur de moins
Si je suis le dernier, réveiller un rédacteur

NL : = NL - 1

Si NL = 0 alors
V(Accès)
fsi

Lecteur

Les sémaphores
Lecteurs / Rédacteurs

NL est accédé en concurrence par tous les lecteurs
Il y a besoin d'une exclusion mutuelle sur son accès

INIT (Accès, 1);
INIT (Mutex, 1);

P(Mutex)
NL : = NL + 1
Si (NL = 1)
alors
P(Accès)
fsi
V(Mutex)

Accès lecture

P(Mutex)
NL := NL - 1;
Si (NL = 0)
alors
V(Accès)
fsi
V(Mutex)

P(Mutex)
NL : = NL + 1
Si (NL = 1)
alors
P(Accès)
fsi
V(Mutex)

LECTURE

P(Mutex)
NL := NL - 1;
Si (NL = 0)
alors
V(Accès)
fsi
V(Mutex)

--> Ecrivain 1
P(Accès)

ECRITURE

V(Accès)

LIBRE, NL = 0

ABCDEFGHIJKLMN

LECTURE (1)

ABCDEFGHIJKLMN

LECTURE (2)

ABCDEFGHIJKLMN

Lecteur 1
Read (fd…)

Si il n'y a pas d'accès en écriture en cours
Accéder au fichier

P(Mutex)
NL = NL + 1 = 1
P(Accès)
V(Mutex)

LIBRE
Accès lecture

Ecrivain 1
Write (fd…)

Si il n'y a pas d'accès en écriture en cours ni en lecture
Accéder au fichier

P(Accès)

Bloqué

Lecteur 2
Read (fd…))

Si il n'y a pas d'accès en écriture en cours
Accéder au fichier

P(Mutex)
NL = NL + 1 = 2
V(Mutex)

Accès lecture

P(Mutex)
NL : = NL + 1
Si (NL = 1)
alors
→(lecteur 1)
P(Accès)
fsi
V(Mutex)

LECTURE

P(Mutex)
NL := NL - 1;
Si (NL = 0)
alors
V(Accès)
fsi
V(Mutex)

→Ecrivain 2
P(Accès)

ECRITURE

V(Accès)

ECRITURE(1)

ABCDEFGHIJKLMN

Lecteur 1
Read (fd…)

Si il n'y a pas d'accès en écriture en cours
Accéder au fichier

P(Mutex)
NL = NL + 1 = 1
P(Accès)

Bloqué

ECRITURE(1)

ABCDEFGHIJKLMN

Ecrivain 2
Write (fd…)

Si il n'y a pas d'accès en écriture en cours ni en lecture
Accéder au fichier

P(Accès);

Bloqué

ECRITURE(1)

ABCDEFGHIJKLMN

Lecteur 2
Read (fd…))

Si il n'y a pas d'accès en écriture en cours
Accéder au fichier

P(Mutex)
NL = NL + 1 = 1
P(Accès)

Bloqué

```
→(lecteur 2)
P(Mutex)
NL : = NL + 1
Si (NL = 1)
alors
→(lecteur 1)
P(Accès)
fsi
V(Mutex)
```

LECTURE

```
P(Mutex)
NL := NL - 1;
Si (NL = 0)
alors
V(Accès)
fsi
V(Mutex)
```

```
→Ecrivain 2
P(Accès)
```

ECRITURE

```
V(Accès)
```

ECRITURE(1)

ABCDEFGHIJKLMN

ECRITURE(1)

ABCDEFGHIJKLMN

ECRITURE() OU LECTURE

ABCDEFGHIJKLMN

Lecteur 2
Read (fd…))

Si il n'y a pas d'accès en écriture en cours
Accéder au fichier

P(Mutex);

Bloqué

Ecrivain 1
Write (fd…)

ACCES ECRITURE

Libérer accès fichier
Réveiller processus en attente

V(Accès);

L'accès suivant est donné au premier processus élu

Ecrivain 2
Write (fd…)

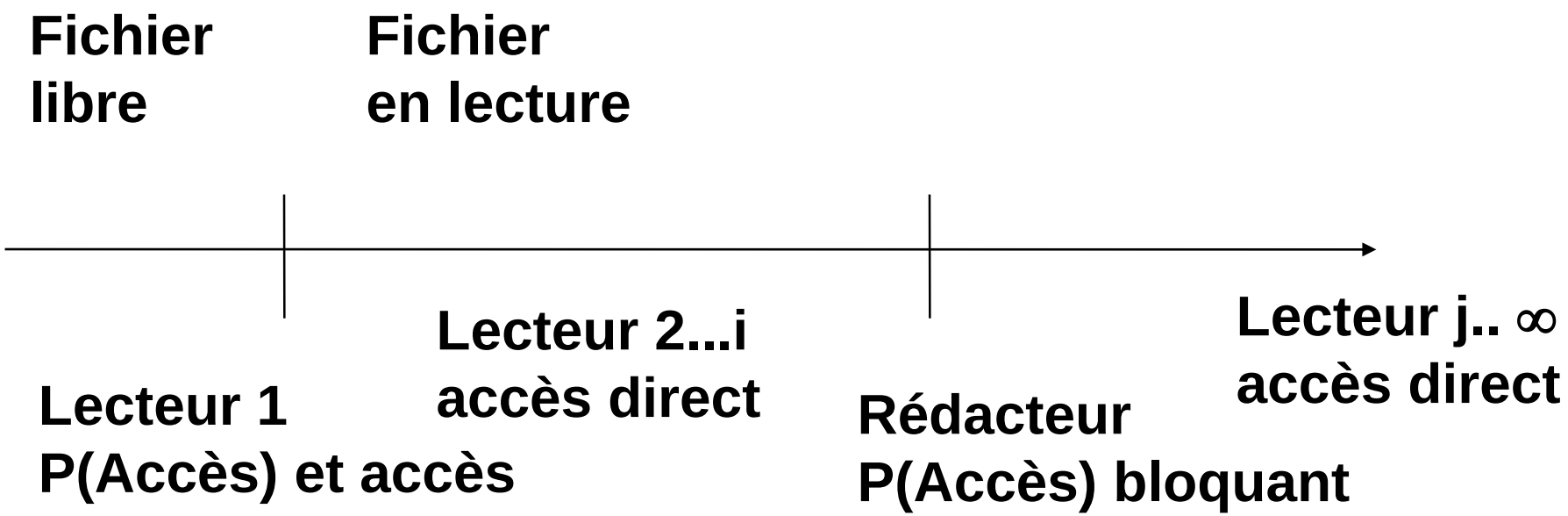
Lecteur 1
Read (fd…))

V(Mutex)

Lecteur 2
Read (fd…))

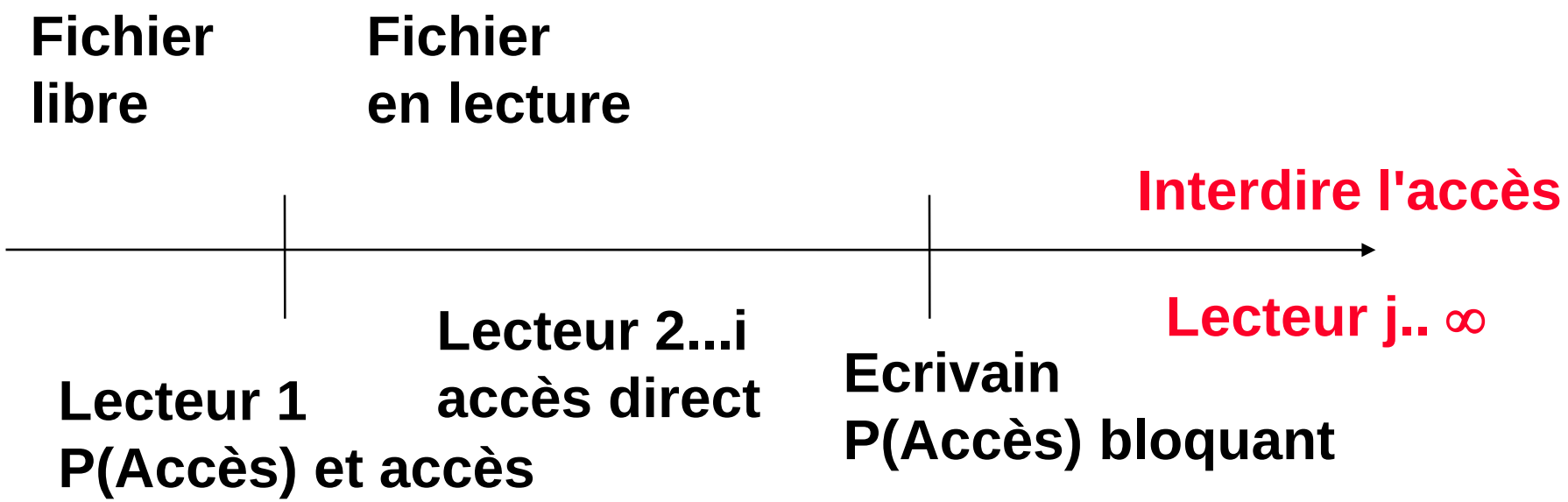
Les sémaphores
Lecteurs / Rédacteurs

- **Coalition des lecteurs contre les rédacteurs**



Les sémaphores
Lecteurs / Rédacteurs

- **Solution à la coalition**



Synchronisation et communication entre processus

Sémaphores Linux

Sémaphores Linux

- Famille des IPCs, un ensemble de sémaphores est identifié par une clef.

- Les opérations P, V et ATT (attente qu'une valeur de sémaphore soit nulle) s'effectuent sur un tableau de sémaphores, atomiquement

  ↳ L'ensemble des opérations est réalisée avant que le processus puisse poursuivre son exécution..

Sémaphores Linux

- Famille des IPCs, un ensemble de sémaphores est identifié par une clef.

- Les opérations P, V et ATT (attente qu'une valeur de sémaphore soit nulle) s'effectuent sur un tableau de sémaphores, atomiquement

  ↳ L'ensemble des opérations est réalisée avant que le processus puisse poursuivre son exécution..

Sémaphores Linux

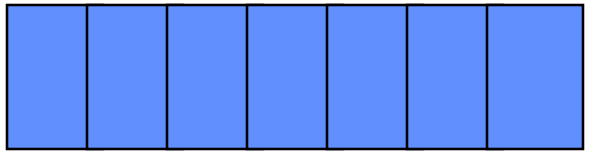
- Création et recherche d'un ensemble de sémaphore

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

Création ou recherche d'un ensemble de n sémaphores identifiés par clé.

Retourne un identifiant interne de type entier

Semflg = constantes IPC_EXCL, IPC_CREAT, 0



nsems

```
Struct semaphore {
atomic_t count ; /* compteur */
int sleepers; /* nombre de processus endormis */
Wait_queue_head_t wait; /* file d'attente */ }
```

Sémaphores Linux

- Opérations sur les sémaphores

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

Réalisation d'un ensemble d'opérations (nsops) décrite chacune dans une structure sembuf sur l'ensemble de sémaphore semid.

```
struct sembuf {
    unsigned short sem_num;/* numéro du sémaphore dans le tableau */
    short sem_op; /* opération à réaliser sur le sémaphore */
    short sem_flg; /* options */
     };
```

– si sem_op est négatif, l'opération à réaliser est une opération P;
– si sem_op est positif, l'opération à réaliser est une opération V;
– si sem_op est nul, l'opération à réaliser est une opération ATT.

Sémaphores Linux

- Initialiser un sémaphore

int semctl (int semid, int semnum, int cmd, union semun arg);

semctl (semid, 0, SETVAL, 3) initialisation à la valeur 3 du sémaphore 0 dans l'ensemble désigné par l'identifiant semid.

- Détruire un ensemble de sémaphores

int semctl (int semid, 0, IPC_RMID, 0);

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

int i, nb_place;
int semid;
struct sembuf operation;

void reservation()
{
/* opération P */
operation.sem_num = 0;
operation.sem_op = -1;
operation.sem_flg = 0;
semop (semid, &operation, 1);
nb_place = nb_place - 1;
/* opération V */
operation.sem_num = 0;
operation.sem_op =  1;
operation.sem_flg = 0;
semop (semid, &operation, 1);
}
```

```c
main()
{ pthread_t num_thread[3];


/* création d'un sémaphore initialisé à
la valeur 1 */
semid = semget (12, 1,
                IPC_CREAT|IPC_EXCL|0600);
semctl (semid, 0, SETVAL, 1);


for(i=0; i<3; i++) {
pthread_create(&num_thread[i], NULL,
(void *(*)())reservation, NULL);
pthread_join(num_thread, NULL);
semctl (semid, 0, IPC_RMID, 0)
}
```