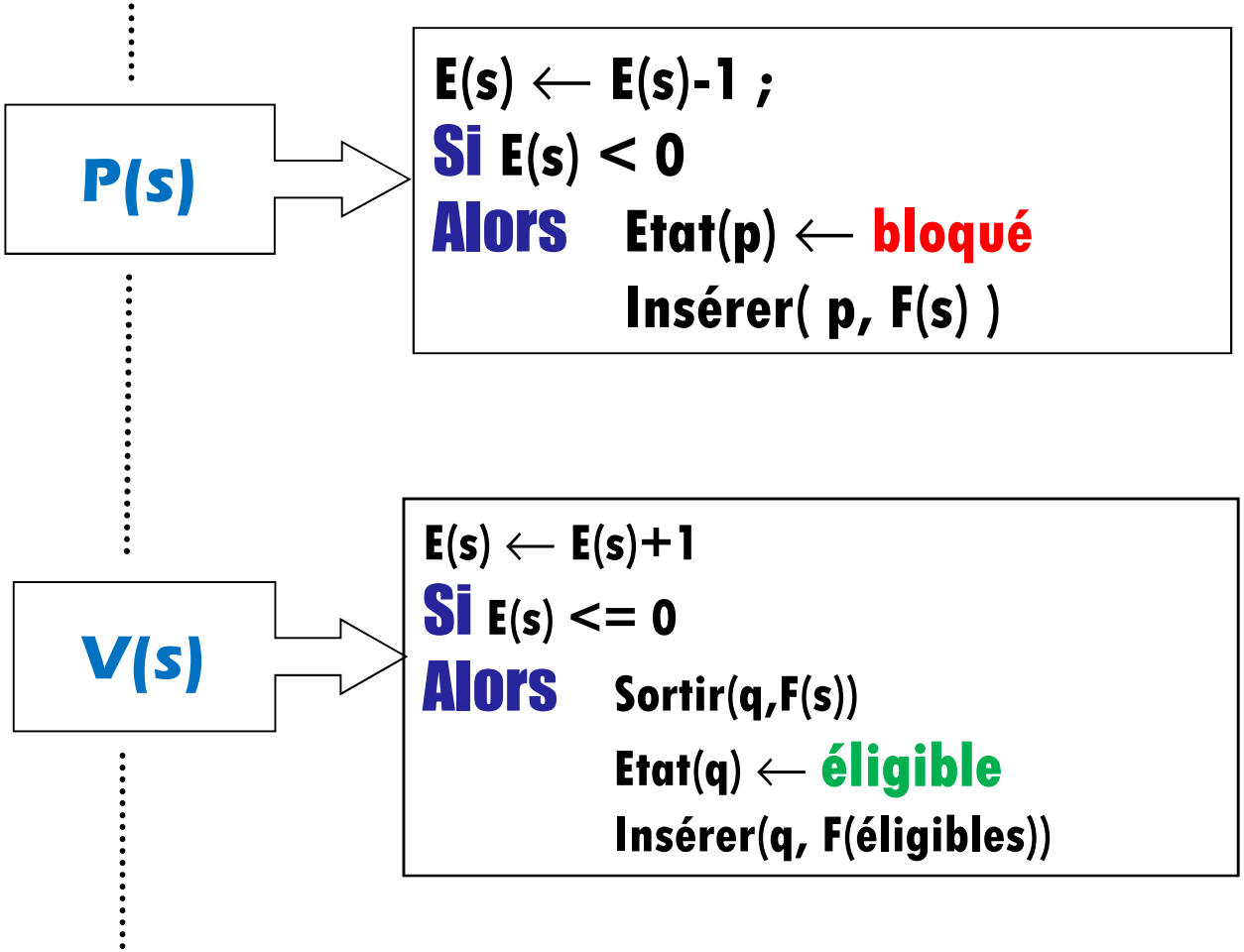
Les sémaphores

Introduits par **Edsger DIJKSTRA** en 1965

Le sémaphore est un outil élémentaire
de synchronisation qui évite l'attente active

Un sémaphore s →
- un entier E(s)
- une file d'attente F(s)
- deux opérations P(s) et V(s)

Processus p


P(s)
E(s) ← E(s)-1 ;
Si E(s) < 0
Alors Etat(p) ← bloqué
Insérer(p, F(s))
V(s)
E(s) ← E(s)+1
Si E(s) <= 0
Alors Sortir(q,F(s))
Etat(q) ← éligible
Insérer(q, F(éligibles))

Le modèle des Lecteurs et des Rédacteurs

Semaphore **mutex=1** ;

Semaphore **Sbd=1**

int Nbl = 0 ;

Processus Lecteur ;

```
TantQue (vrai)
        P(mutex) ;
        Nbl = Nbl+1 ;
        Si (Nbl == 1) P(Sbd) ;
        V(mutex) ;
        Lire_Base_de_Données ;
        P(mutex) ;
        Nbl = Nbl-1 ;
        Si (Nbl == 0) V(Sbd) ;
        V(mutex) ;
        Utiliser_Données_Lues ;
FinTantQue
```

Processus Rédacteur ;

```
TantQue (vrai)
        Créer_Données ;
        P(Sbd) ;
        Ecrire_Données ;
        V(Sbd) ;
FinTantQue
```

Modèle LECTEURS - REDACTEURS

Soit un fichier manipulé par 2 catégories de processus

Les LECTEURS qui n'y accèdent qu'en LECTURE
Les REDACTEURS qui y accèdent en ECRITURE

Afin de préserver la cohérence du fichier,
nous admettons que le comportement des processus obéit aux règles suivantes

Les LECTEURS peuvent lire à plusieurs
à condition qu'il n'y ait pas de rédaction en cours

A un moment donné, on ne peut avoir qu'un seul rédacteur en train d'écrire.

Structure des processus

Processus LECTEUR

```
while (1) {
        DebutLecture( ) ;
        Lire() ;
        FinLecture() ;
}
```

Fin du Processus

Processus REDACTEUR

```
while (1) {
        DebutRedaction() ;
        Ecrire() ;
        FinRedaction() ;
}
```

Fin du Processus

NbLecteurs

variable contenant le nombre de lecteurs en cours

Mutex

Sémaphore d'exclusion mutuelle sur **NbLecteurs**

Lr

Sémaphore d'exclusion mutuelle entre lecteurs et rédacteurs

R

Sémaphore de protection de la ressource (fichier)

Priorités EGALES

```
DebutLecture() {
        P(Lr) ;
        P(Mutex);
        NbLecteurs ++ ;
        if (NbLecteurs == 1) P(r) ;
        V(Mutex) ;
        V(Lr) ;
}
```

```
FinLecture() {
        P(Mutex);
        NbLecteurs - - ;
        if (NbLecteurs == 0) V(r) ;
        V(Mutex) ;
}
```

```
DebutRedaction() {
        P(Lr) ;
        P(r) ;
}
```

```
FinRedaction() {
        V(r) ;
        V(Lr) ;
}
```

Priorité aux LECTEURS

```
DebutLecture() {
        P(Mutex);
        NbLecteurs ++ ;
        if (NbLecteurs == 1) P(r) ;
        V(Mutex) ;
}
```

```
FinLecture() {
        P(Mutex);
        NbLecteurs - - ;
        if (NbLecteurs == 0) V(r) ;
        V(Mutex) ;
}
```

```
DebutRedaction() {
        P(r) ;
}
```

```
FinRedaction() {
        V(r) ;
}
```

Priorité aux REDACTEURS

```
DebutLecture() {
        P(p_rl) ;
        P(scl) ;
        P(Mutex);
        NbLecteurs ++ ;
        if (NbLecteurs == 1) P(r) ;
        V(Mutex) ;
        V(scl) ;
        V(p_rl) ;
}
```

```
FinLecture() {
        P(Mutex);
        NbLecteurs - - ;
        if (NbLecteurs == 0) V(r) ;
        V(Mutex) ;
}
```

```
DebutRedaction() {
        P(Mutex) ;
        NbRedacteurs ++;
        if (NbRedacteur == 1) P(scl) ;
        V(Mutex) ;
        P(r) ;
}
```

```
FinRedaction() {
        V(r) ;
        P(Mutex) ;
        NbRedacteurs - - ;
        if (NbRedacteurs == 0) V(scl) ;
        V(Mutex) ;
}
```

Les sémaphores

Création d'un sémaphore

un utilisateur doit lui associer une clé.
Retour : un identificateur auquel sont attachés n sémaphores
(ensemble de sémaphores),
numérotés de 0 à n-1.

Pour spécifier un sémaphore,

l'utilisateur devra alors indiquer l'identificateur de sémaphore
et le numéro de sémaphore.

Les sémaphores

A chaque sémaphore est associée une valeur, **toujours positive**,
qu'on peut **incrémenter** ou **décrémenter** d'un nombre quelconque.

N : la valeur initiale,
n : le nombre d'incrémentation de l'utilisateur

si n > 0 alors on **augmente** la valeur du sémaphore de **n**
et on **continue en séquence**.

si n < 0
si N+n ≥ 0 alors on **diminue** la valeur du sémaphore de **|n|**
et on **continue en séquence**.

si N+n < 0 alors le **processus se bloque**, en attendant que **N+n≥0**.

si n==0
si N==0 alors l'utilisateur **continue en séquence**.
si N≠0 alors l e processus **se bloque** en attendant que **N=0**.

Les sémaphores

Les opérations effectuées sur les sémaphores sont atomiques – in-interruptibles,

S'exécutent toujours séquentiellement même sur une machine multiprocesseurs.

Les sémaphores

Création d'un nouvel ensemble de sémaphores

int semget(key_t key, int nsems, int semflg)

crée un nouvel ensemble de sémaphores
ou pour **obtenir l'identificateur** de sémaphore d'un ensemble existant.

key >0 (IPCPRIVATE(=0))
est une clé indiquant le nom numérique de l'ensemble de sémaphores.

nsems indique le nombre de sémaphores de l'ensemble.
semflg est un flag spécifiant les droits d'accès.

La valeur retournée par **semget()** est égale à **-1** en cas d'échec.

Les sémaphores

Exemple – création d'un ensemble de sémaphores

Ce programme crée un ensemble de quatre sémaphores associé à la clé 123.

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

#define CLE 123
int main( ) {
  int semid ;

  /* Création de 4 sémaphores */
  if ( ( semid = semget( (key_t)CLE, 4, IPC_CREAT | IPC_EXCL | 0666)) == -1) {
            perror("Echec de semget") ;
            exit(1) ;
  }
  printf(" le semid de l'ensemble de semaphore est : %d\n",semid) ;
  printf(" cet ensemble est identifie par la cle unique : %d\n" , (key_t)CLE) ;
  return 0;
}
```

```
$ test_semget
le semid de l'ensemble de semaphore est : 2
cet ensemble est identifie par la clé unique : 2073103658
$ ipcs
IPC status from /dev/kmem as of Fri Jun 21 11:08:06 1999
T ID KEY MODE OWNER GROUP
Message Queues:

Shared Memory:
m 100 0x0000004f --rw-rw-rw- root root
Semaphores:
s 2 0x7b910d2a --ra------- toto speinf
$ test_semget
Echec de semget: File exists
```

Les sémaphores

Consultation et changement des valeurs de sémaphores d'un ensemble de sémaphores.

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

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

union semun {
   int val ;
   struct semid_ds *buf ;
   ushort array[] ;     /*tableau de taille égale au nombre de sémaphores de l'ensemble */
} arg ;
```

La valeur retournée dépend de la valeur de l'argument *cmd* :

si *cmd* = GETVAL : valeur de semval

si *cmd* = GETPID : valeur de sem_pid

si *cmd* = GETNCNT : valeur de sem_ncnt

si *cmd* = GETZCNT : valeur de sem_zcnt

Les sémaphores

Exemple – Utilisation de la fonction semctl()

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

#define CLE 123
union semun {
   int val ;
   struct semid_ds *buf ;
   ushort array[4] ;
} arg ;

main() {
   int semid, val_sem ,val_pid;
   /* récupération de l'identificateur de l'ensemble de sémaphores de clé 123 */
   semid = semget(key_t)CLE,4,0)
   printf("L'ensemble de semaphore a comme semid : %d\n",semid) ;
   printf("La clé d'acces est %d\n",(key_t)CLE) ;
   arg.val = 1 ;
   semctl(semid, 2, SETVAL, arg) == -1) ;    /* mise à 1 du 3ème sémaphore */
   val_sem = semctl(semid, 2, GETVAL, arg) ;          /*lecture du 3ème sémaphore */
   printf("la valeur du troisieme semaphore est : %d\n",val_sem) ;
  /* lecture du pid du processus qui a effectue la dernière opération */
   val_pid = semctl(semid,2,GETPID,arg) ;
   printf("la valeur du pid du processus qui a effectue la dernière opération est : %d,\n mon pid est
:%d\n",val_pid,getpid()) ;
   semctl(semid,0,IPC_RMID,0)      /* destruction du semaphore */
}
```

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

```
#define CLE 123     /* fichier processus1.c - programme exécuté par le premier processus */
int semid ;
struct sembuf operation[1] ;
 union { int val ; struct semid_ds *buf ;  ushort array[4] ; } arg;

main() {    /* création d'un ensemble de 4 sémaphores */
   semid = semget( (key_t)CLE, 4, IPC_CREAT | SEM_A | SEM_R) ;
   printf("process1: Création d'un ensemble de sémaphore : %d\n", semid) ;
   arg.val=1 ;          /* mise  à 1 du troisième sémaphore */
   semctl(semid,2,SETVAL,arg) ;
   printf("process1: je vais demander une ressource \n") ;
   operation[0].sem_num = 2 ;  /*opération sur le troisième sémaphore */
   operation[0].sem_op = -1 ;       /*opération de décrémentation */
   operation[0].sem_flg = 0;      /* pour défaire les opérations*/
   semop(semid,operation,1)
    printf("process1: j'attends 10 secondes \n") ;  /*attente pour bloquer le processus2 */
    sleep(10) ;          /* attente ... */
    operation[0].sem_op = 1 ;                          /* Incrémentation */
    semop(semid,operation,1)
    printf("mort de process1\n") ; exit(0) ;
}
```

/* fichier **processus2.c** - programme exécuté par le second processus */

```
#define CLE 123
int semid ;
struct sembuf operation[1] ;

int main() {
   semid = semget((key_t)CLE,0,0);          /* récupération du semid */
   printf("process2: traite les sémaphore : semid %d\n",semid) ;
   /* boucle d'attente de la disponibilité du sémaphore.
    * On demande de ne pas rester bloqué en attente
    * en positionnement le drapeau IPC_NOWAIT */

    operation[0].sem_num = 2 ;
    operation[0].sem_op = -1 ;
    operation[0].sem_flg = IPC_NOWAIT  ;
    for (;;){
        if ( semop(semid,operation,1) != -1) break ;
        printf(" demande du process2 : sémaphore non disponible \n") ;
        sleep(1) ;
    }
    printf(" sémaphore alloué au process2\n") ;
    semctl(semid,0,IPC_RMID,0);              /* libération du sémaphore */
    return 0;
}
```

Les sémaphores

Implémentation des sémaphores de Dijkstra

```
int sem_create(key_t  cle, int  initval)  {                    /* fichier dijkstra.c */
  int semid ;
  union semun { int val ; struct semid_ds *buf ; ushort *array ; } arg_ctl ;

  semid = semget(cle, 1 , IPC_CREAT|IPC_EXCL|0666) ;
  if (semid == -1) {
            semid = semget(cle,1, 0666) ;
            if (semid == -1) {
                        perror("Erreur semget()") ; exit(1) ;
            }
  }
  else {    arg_ctl.val = initval ;
            if (semctl(semid, 0, SETVAL, arg_ctl) == -1) {
                        perror("Erreur initialisation sémaphore") ; exit(1) ;
            }
  }
  return(semid) ;
}
```

Les sémaphores

Implémentation des sémaphores de Dijkstra

```
void P(int semid) {
        struct sembuf sempar ;
        sempar.sem_num = 0 ;
        sempar.sem_op = -1 ;
        sempar.sem_flg = 0 ;
        if (semop(semid, &sempar, 1) == -1)  perror("Erreur operation P") ;
}

void V(int semid) {
        struct sembuf sempar ;
        sempar.sem_num = 0 ;
        sempar.sem_op = 1 ;
        sempar.sem_flg = 0 ;
        if (semop(semid, &sempar, 1) == -1)  perror("Erreur opération V") ;
}

 void sem_delete(int semid) {
        if (semctl(semid,0,IPC_RMID,0) == -1)  perror("Erreur dans destruction sémaphore") ;
}
```

Les sémaphores

Exemple d'utilisation des sémaphores de Dijkstra

```
#include <sys/types.h>                  /*fichier test_sem_dijkstra.c */
#include <sys/ipc.h>
#include <sys/sem.h>
#include "dijkstra.h"
#define CLE 1
main() {
        int sem ;
        sem = sem_create(CLE,0) ;
        printf("Creation du sémaphore d'identificateur %d\n",sem) ;
        if (fork() == 0) {
                printf("Je suis le fils et j'attends 15 secondes...\n") ; sleep(15) ;
                printf("Je suis le fils et je fais V sur le sémaphore\n") ; V(sem) ;
                exit(0) ;
        }
        else {
                printf("Je suis le père et je me bloque en faisant P sur le sémaphore\n\n") ;
                P(sem) ;
                printf("Je suis le père et je suis libre\n\n") ;
                sem_delete(sem) ;
        }
}
```