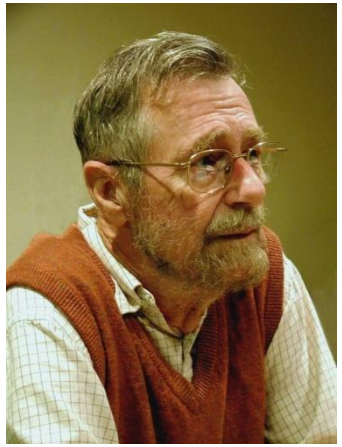
# LINUX – LE NOYAU
# LES SEMAPHORES

# Edgser Dijkstra

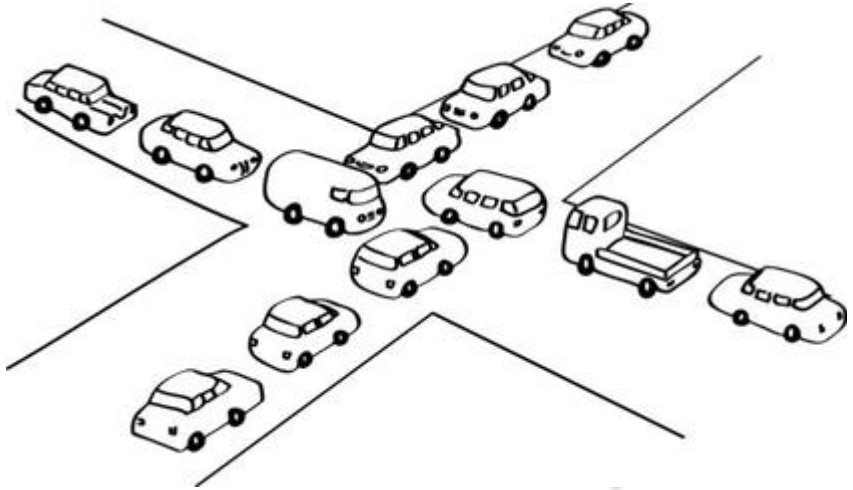
Mathématicien et informaticien néerlandais
(1930 – 2002)



- Algorithme de calcul du plus court chemin dans les graphes (Algorithme de Dijkstra - 1959)
- **The operating system** : système construit en couches d'abstraction successives
  -> Formalisation de la notion de sémaphores (1965)

# Notion d'interblocage (deadlock)

- Phénomène pouvant se produire lorsque des activités s'effectuent de façon concurrente
  - Cela arrive très souvent en informatique (programmation concurrente)
- L'interblocage se produit lorsqu'au moins deux acteurs (processus) s'attendent mutuellement
  - Les acteurs bloqués dans cet état le sont définitivement ce qui peut être catastrophique en informatique



# Notion de section critique

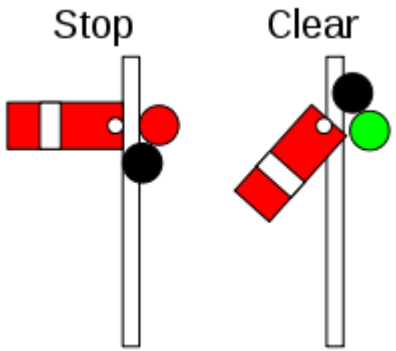
- On appelle section critique la partie d'un programme où il y a accès concurrent à une même donnée
  - Un espace mémoire sur un serveur
  - Voire même… un carrefour routier, une section unique de voie ferrée…

- Dans une section critique, il doit être garanti qu'il n'y aura jamais plus d'un « processus » simultanément
  - Objectif : Éviter les conflits d'accès tout en garantissant une équité d'accès

# Notion de sémaphore

- Concept introduit en 1960 par Edgser Dijkstra

- Mécanisme très utilisé dans les systèmes d'exploitation multitâches :
  - Contrôler l'accès à une ressource partagée
    - Exclusion mutuelle (mutex)
  - Signaler des événements
  - Synchroniser des tâches

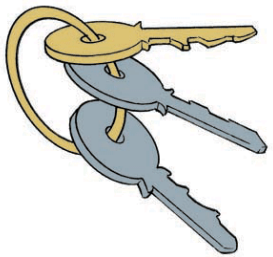


# Principe du sémaphore

**Sémaphore = clé pour accéder à une ressource**

- Si je veux accéder à la ressource

  **je demande la clé**

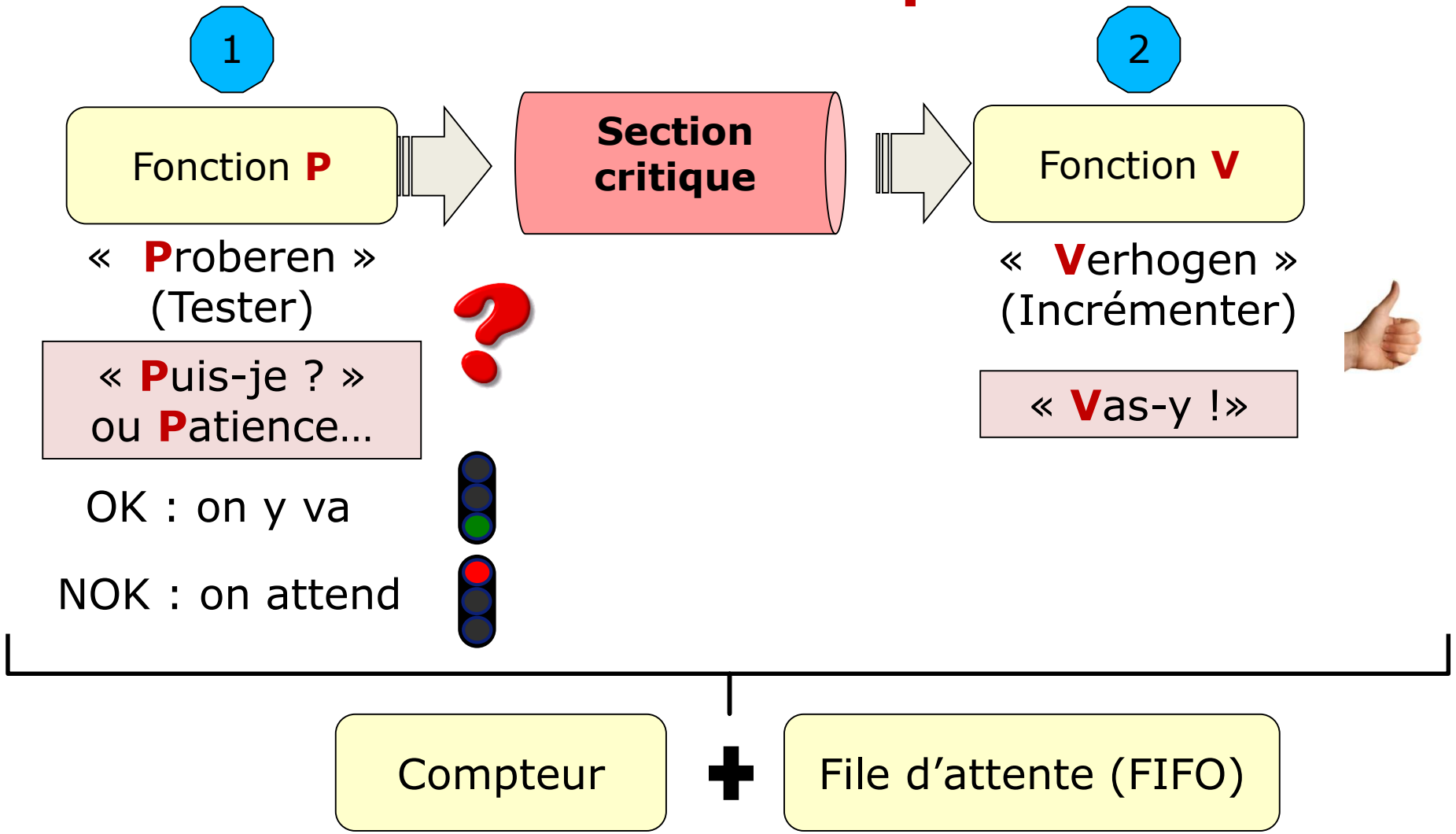
- J'accède à la ressource puis

  **je rends la clé**

- Si la clé est prise

  **j'attends la ressource pour y accéder**

# Gestion du sémaphore

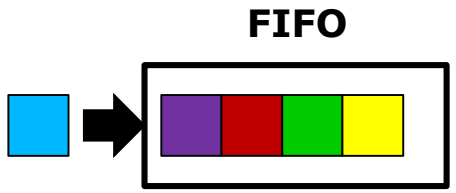


# Représentation en Pseudo-langage

```
Initialiser
    Cpt ← N
```

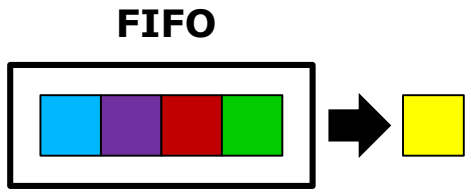
Initialisation du compteur

```
P
    Cpt ← Cpt – 1;
    si Cpt < 0 alors
        Enfiler(tâche);
        Suspendre(tâche);
```



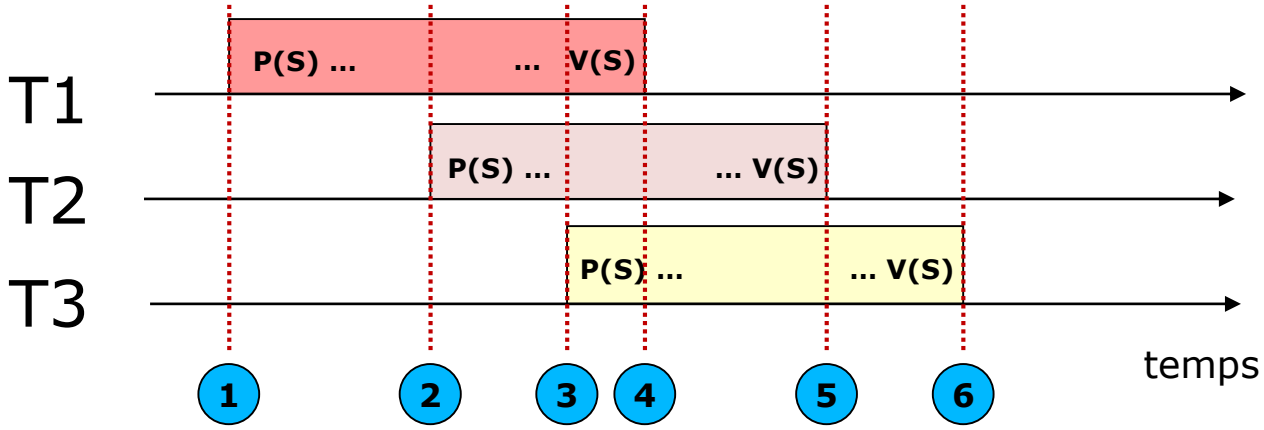

Puis-je ?

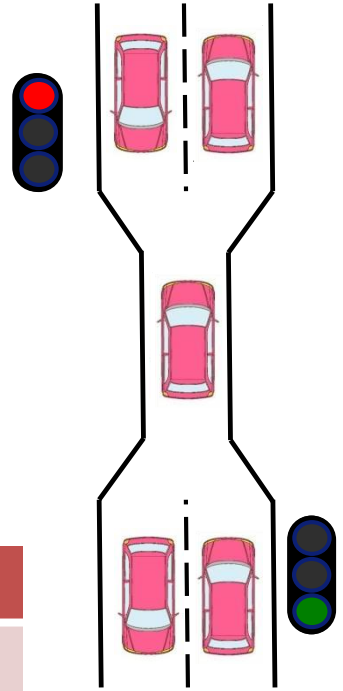
```
V
    Cpt ← Cpt + 1;
    Défiler(tâche);
    Réveiller(tâche);
```




Vas-y !

# Exemple avec tois acteurs



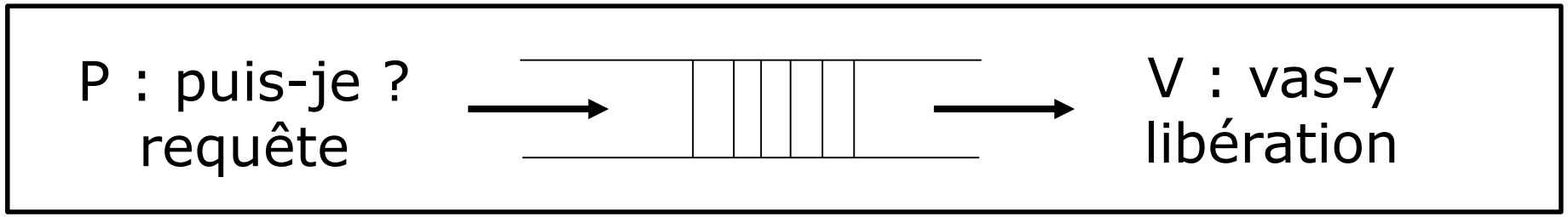



| | Init | S=1 | F=Vide |
|---|---|---|---|
| 1 | T1(P) | 0 | Vide |
| 2 | T2(P) | -1 | T2 |
| 3 | T3(P) | -2 | T2,T3 |
| 4 | T1(V) | -1 | T3 |
| 5 | T2(V) | 0 | Vide |
| 6 | T3(V) | 1 | Vide |

# Représentation

Sémaphore = file (FIFO) associée à un compteur

File où sont enregistrés les identifiants des tâches en attente de la ressource




Compteur : nombre de ressources identiques disponibles (mutex)
Compteur < 0 : pas de ressource disponible
Compteur >= 0 : une (ou +) ressource(s) disponible(s)

# Communications inter-processus

- Problématique
  - Assurer le partage des ressources
  - Gérer la concurrence pour l'utilisation des ressources
  - Garantir la synchronisation des processus

- Exemple de communications
  - Exclusion mutuelle (verrou)
  - Synchronisation
  - Rendez-vous
  - Interruptions
  - …

- Des outils…
  - Les sémaphores et leur dérivés

# Exclusion mutuelle (mutex)

- Objectif
  - Permet d'éviter que des ressources partagées d'un système ne soient utilisées en même temps
    - Ex : variables globales, fichiers
- Principe

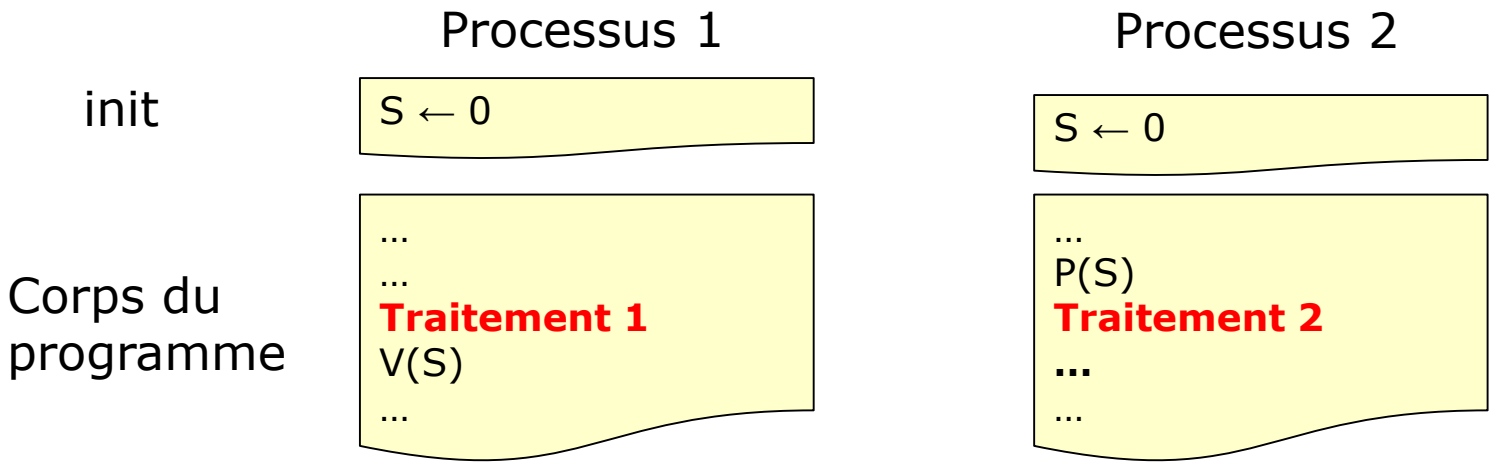
| | Processus 1 | Processus 2 |
|---|---|---|
| init | S ← Nb_ressources | S ← Nb_ressources |
| Corps du programme | …<br>P(S)<br>**Section critique**<br>V(S)<br>… | …<br>P(S)<br>**Section critique**<br>V(S)<br>… |

- Attention
  - L'algorithme parfait n'existe pas
  - Tous sont faillibles dans des conditions données (interblocage)

# Synchronisation

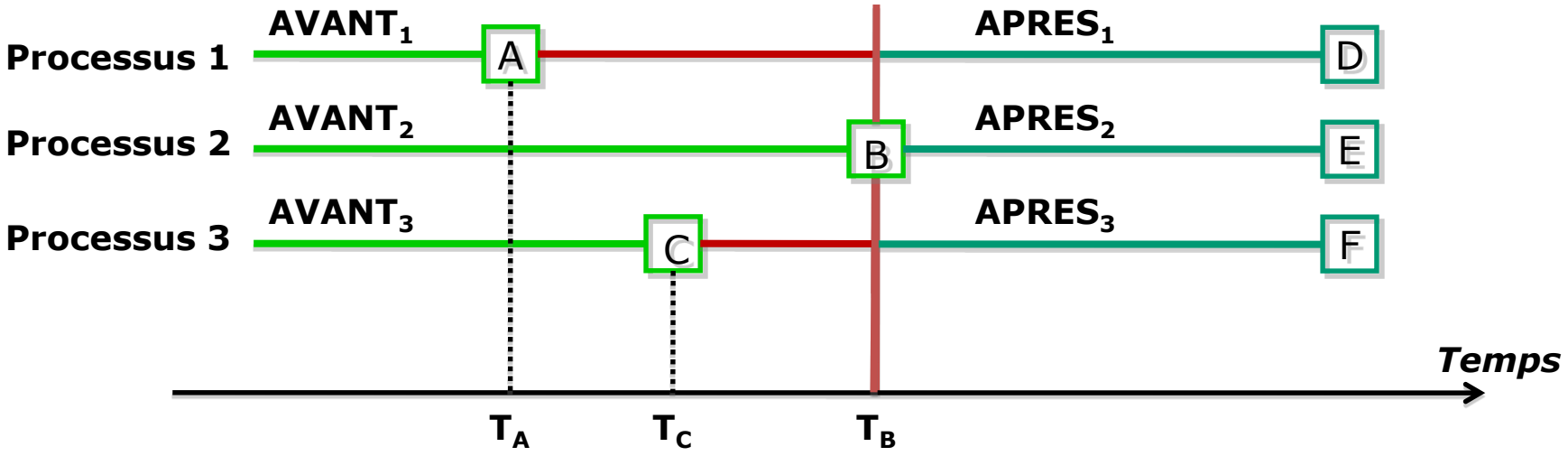
- Objectif
  - Un processus doit en attendre un autre pour continuer (ou commencer) son traitement

- Principe



# Problème du rendez-vous

- Pour chaque processus : phase AVANT et phase APRES
- Problème : garantir qu'aucun processus ne commence sa phase de calcul APRES avant que tous les participants aient terminé leur phase de calcul AVANT



# Rendez-vous

- Objectif
  - Un **processus X** doit attendre que *n* autres processus aient terminé un traitement donné pour poursuivre son exécution
- Principe

Initialisation

```
Synchro ← 0
Attente ← 0
Mutex ← 1

Entier Var ← 0
```

Processus i (i = 1 à n)

```
...
Traitement AVANT
...
P(Mutex)
  Var ← Var + 1;
  Si Var = n alors
    V(Synchro)
    Var ← 0
  Fin Si
V(Mutex)
P(Attente)
...
Traitement APRES
...
```

Processus X

```
...
Traitement AVANT
...

P(Synchro)
V(Attente, n)

...
Traitement APRES
...
```

# En résumé…

- En **programmation concurrente**, se posent des problèmes **d'accès à des ressources partagées** et des problématiques de **synchronisation**
  - le risque est que les processus se retrouvent en interblocage
- Le **sémaphore** est un outil de base servant à résoudre de **façon déterministe** des problèmes :
  - Exclusion mutuelle
  - Synchronisation de processus
  - Rendez-vous…

- Un sémaphore comporte deux primitives :
  - Une fonction P() : demande d'autorisation
  - Fonction V() : fin d'utilisation

# Threads et sémaphores

- Standard POSIX 1003.1b
  (Portable Operating System Interface)
  - La librairie de gestion des threads offre des fonctions pour créer et utiliser des sémaphores

- Attention : ces sémaphores sont propres à un processus
  - Ils permettent de synchroniser plusieurs threads entre eux mais ils ne peuvent synchroniser plusieurs processus

  - Pour réaliser cette synchronisation il faut se tourner vers les sémaphores système V basés sur les IPC (Inter Processus Communication)

# Fonctions de gestion des sémaphores

```
#include <pthread.h>
#include <semaphore.h>


int sem_init( sem_t *semaphore, int pshared, unsigned int valeur )
```

Création d'un sémaphore et préparation d'une valeur initiale
**pshared** indique si le sémaphore est local au processus courant (pshared vaut zéro) ou partagée entre plusieurs processus (pshared non nul)
sem_init renvoie toujours l'erreur **ENOSYS** si pshared n'est pas nul

# Fonctions de gestion des sémaphores

| | |
|---|---|
| **int sem_wait (sem_t *semaphore)** | Opération P sur un sémaphore |
| **int sem_trywait (sem_t *semaphore)** | Version non bloquante de l'opération P sur un sémaphore |
| **int sem_post (sem_t *semaphore)** | Opération V sur un sémaphore |
| **int sem_getvalue (sem_t *semaphore, int *sval)** | Récupérer le compteur d'un sémaphore |
| **int sem_destroy (sem_t *semaphore )** | Destruction d'un sémaphore |

# Exemple

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

sem_t sem;

void* func(void* arg){
  int* p = (int*)arg;
  int i,j;
  for(i=0;i<5;i++){
    sem_wait(&sem);
    for(j=0;j<5;j++){
      printf("%d",*p);
    }
    printf("\n");
    sched_yield();
    sem_post(&sem);
    sleep(1);
  }
}
```

```c
int main ( ) {
  int i=0;
  pthread_t th[2];
  int tab[2];
  sem_init(&sem,0,1);
  for (i=0;i<2;i++){
   tab[i] = i+1;
   pthread_create(&th[i], NULL, func,(void*)(tab+i));
  }
  for (i=0;i<2;i++){
   pthread_join(th[i], NULL);
  }
  return 0;
}
```

```
>>./a.out
22222
11111
22222
...
```

gcc votre_programme.c **-lpthread** -o votre_executable

# Exemple