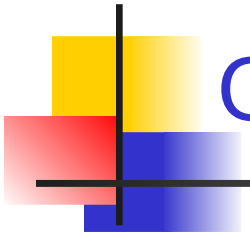
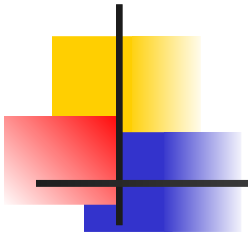


Luc Courtrai

CONCURRENCE

Université de Bretagne SUD
UFR SSI - Departement MIS



Concurrence

Plan

Notion de processus

Les appels système des processus UNIX

La synchronisation entre processus

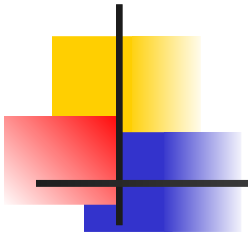
La communication entre processus

Les threads JAVA

Les Posix threads

Concurrence/synchronisation/signaux

- Problème de synchronisation
- TestEtSet
- Sémaphores
- Monitors
- Segments partagés (mémoire)
- Signaux
-



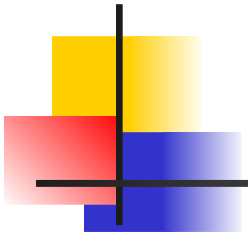
Concurrence/synchronisation

La synchronisation de processus

Sur une machine, les processus utilisent des ressources communes (matérielles ou systèmes : exemple la mémoire, un fichier où chacun peut y lire et écrire.

Problème :

- conflits d'accès à une ressource
- interblocage (dead lock)
- famine



Concurrence/synchronisation

La synchronisation de processus

Prenons l'exemple d'un spool d'impression où les processus clients demandent l'impression de fichiers.

Imaginons le protocole suivant :

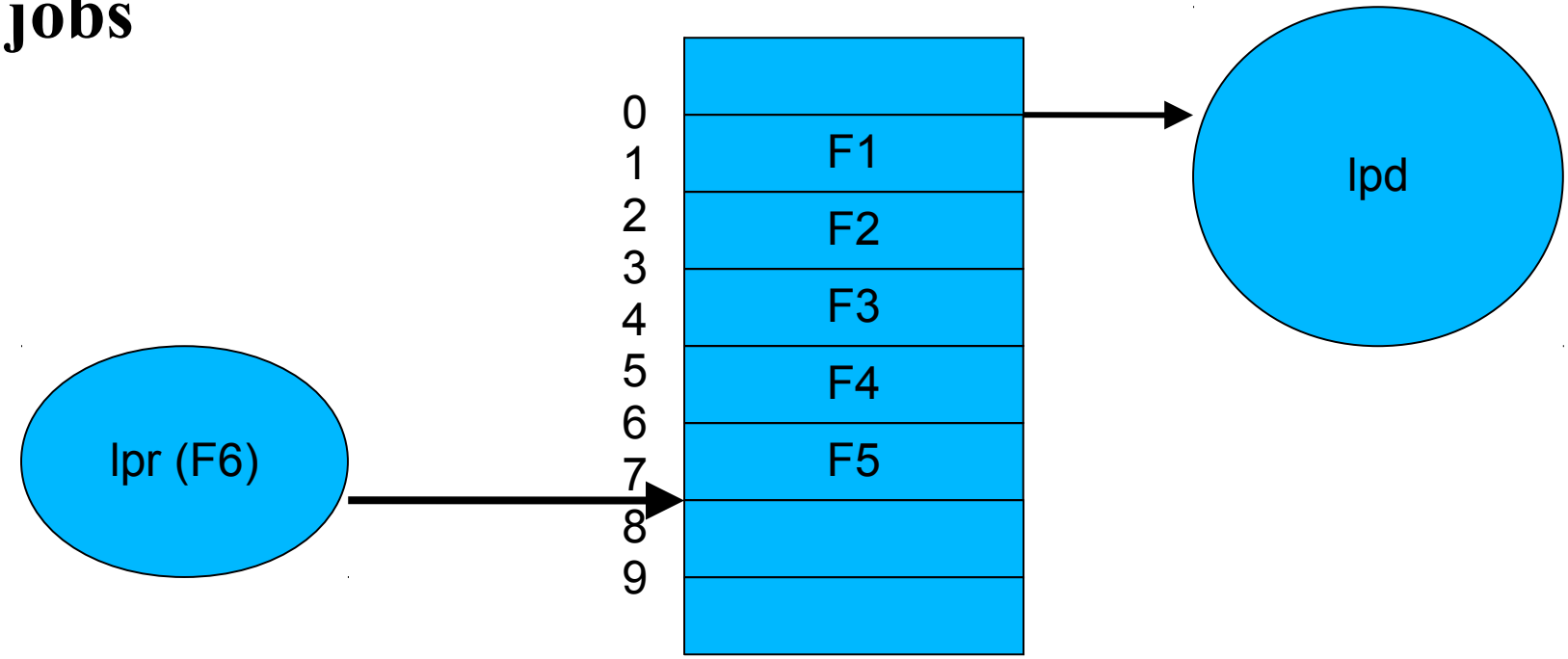
Les processus clients déposent leur fichiers dans un répertoire spécial et inscrit le jobs dans une table.

Un processus "daemon" vérifie périodiquement s'il faut imprimer un fichier.

Concurrence/synchronisation

La synchronisation de processus

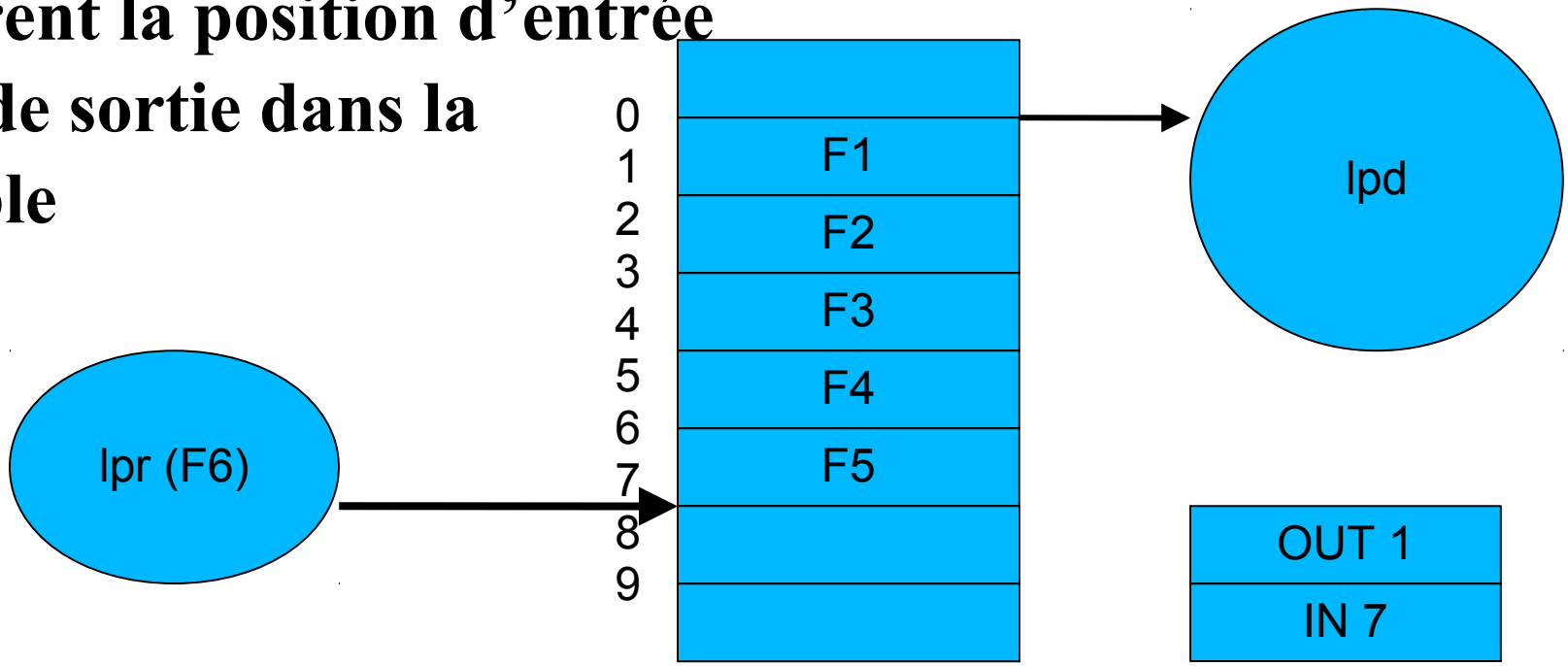
Le système gère une table de jobs

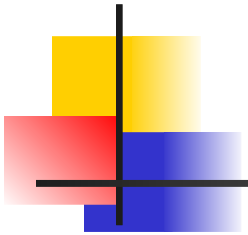


Concurrence/synchronisation

La synchronisation de processus

deux variables partagées gèrent la position d'entrée et de sortie dans la table



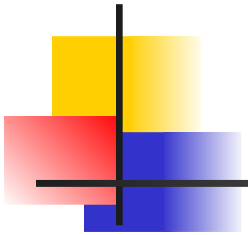


Concurrence/synchronisation

La synchronisation de processus

Le code du serveur pourrait être le suivant

```
while(1)
  if (out < in) {
     imprimer("/var/spool/lpq",table[out])
     out ++
   }
}
```

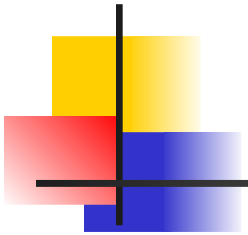


Concurrence/synchronisation

La synchronisation de processus

Le code d'un client pourrait être le suivant

```
{
copie("/var/spool/lpq",file);
strcpy(table[in],file);
in++;
}
```



Concurrence/synchronisation

La synchronisation de processus

Prenons l'exemple de deux processus clients en parallèle voulant imprimer leur fichier.

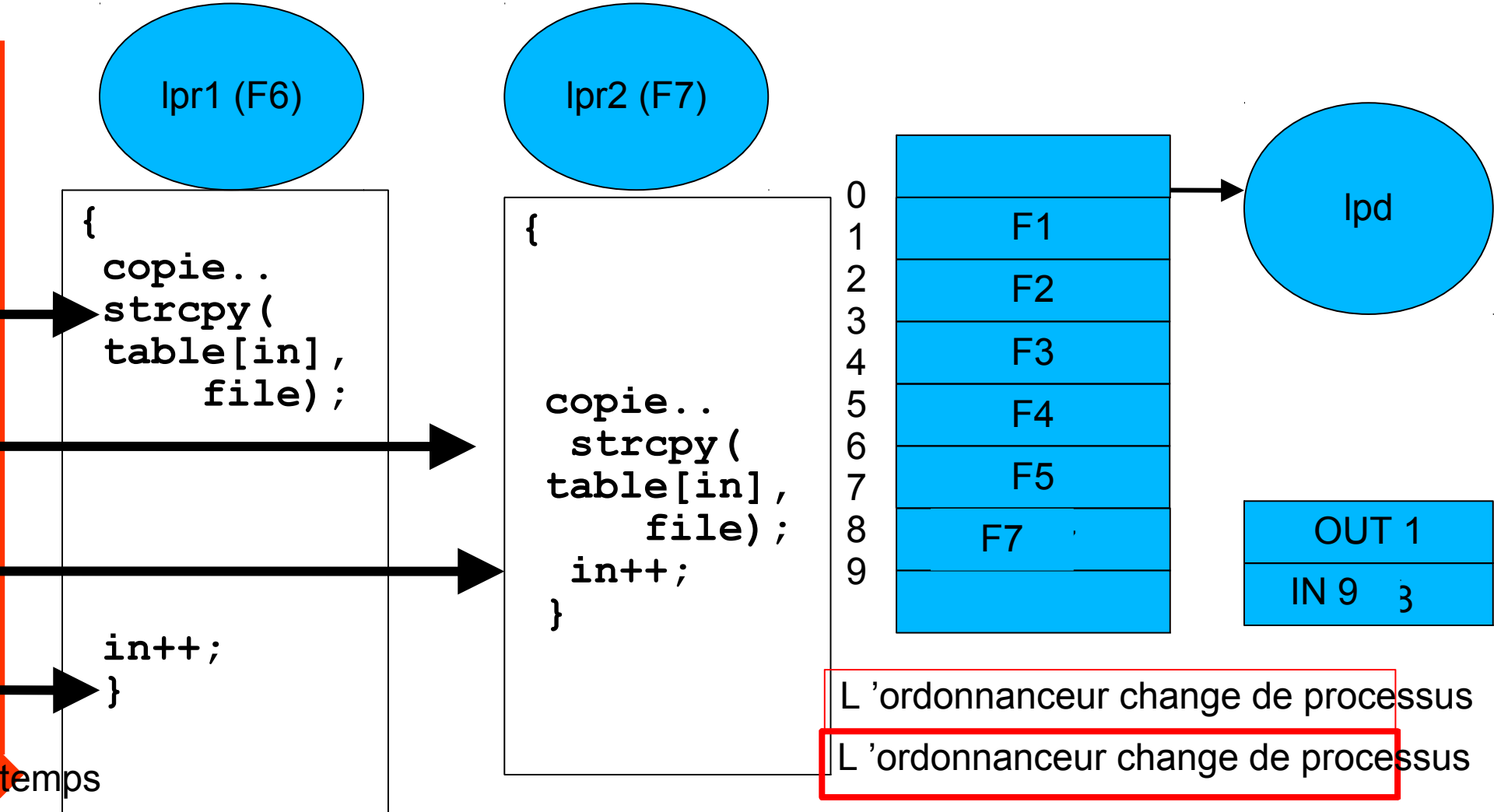
> lpr1 F6

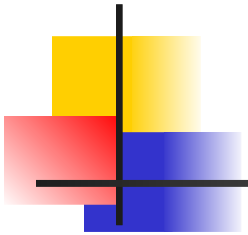
et

> lpr2 F7

Concurrence/synchronisation

La synchronisation de processus





Concurrence/synchronisation

La synchronisation de processus

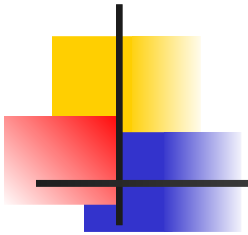
Le système ne fonctionne pas, on a perdu une entrée dans la table et la valeur de la variable IN est fausse.

On constate que la table et la variable IN sont particulières

Il y des **accès concurrents** à la table et à la variable

Et on voudrait protéger des bouts de code.

```
strcpy(table[in],file;
in++;
```



Concurrence/synchronisation

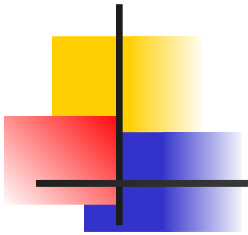
La synchronisation de processus

Section critique

Le bout de code est une **section critique**

```
strcpy(table[in],file;
in++;
```

Le système doit proposer des outils par garantir l ’**exclusion mutuelle** des processus dans une section critique

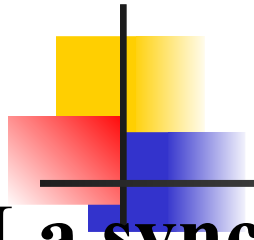


Concurrence/synchronisation

La synchronisation de processus

Pour garantir le bon fonctionnement quatre hypothèses doivent être vérifiées :

- Deux processus ne peuvent être en même temps dans la même section critique
- Aucune hypothèse ne doit être faite sur les vitesses relatives des processus et sur le nombre de processeurs
- Aucun processus suspendu en dehors d'une section critique ne doit bloquer un autre processus qui est en section critique
- Aucun processus ne doit attendre trop longtemps avant d'entrer en section critique



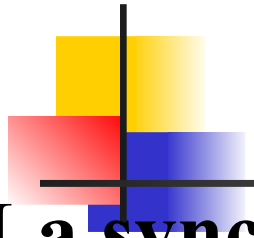
Concurrence/synchronisation

La synchronisation de processus

De plus dans le code du serveur :

```
while(1)
   if (out < in) {
      imprimer("/var/spool/lpq",table[out])
      out ++
   }
}
```

on constate que si il n'y a aucun job, le serveur consomme de la CPU inutilement. C'est de l'attente active.



Concurrence/synchronisation

La synchronisation de processus

```
while(1)
  if (out < in) {
     imprimer("/var/spool/lpq",table[out])
     out ++
  } else
     sleep(1)
}
```

Ce genre de modification consomme aussi de la CPU.

On aimerait avoir des primitives **bloquantes** qui permettent de bloquer un processus ; une autre primitive permettant de le réveiller (par exemple les clients lpr débloque le serveur d'impression).

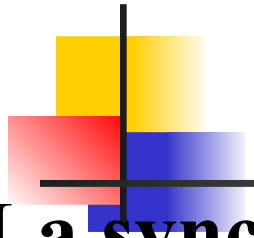
La synchronisation de processus : outils

testAndSet

TestAndSet est une opération spéciale, elle permet de tester et positionner (test and Set) une variable sans être interrompu. Les deux actions sont indivisibles.

C'est OS qui peut garantir l'atomicité cette instruction.

```
int testAndSet(int *b) {
   tmp  = b ;
   *b = 1;
   return tmp;
}
```



La synchronisation de processus : outils

testAndSet

exemple d ’exclusion mutuelle avec le testAndSet

une variable verrou est initialisée à 0;

verrou = 0;

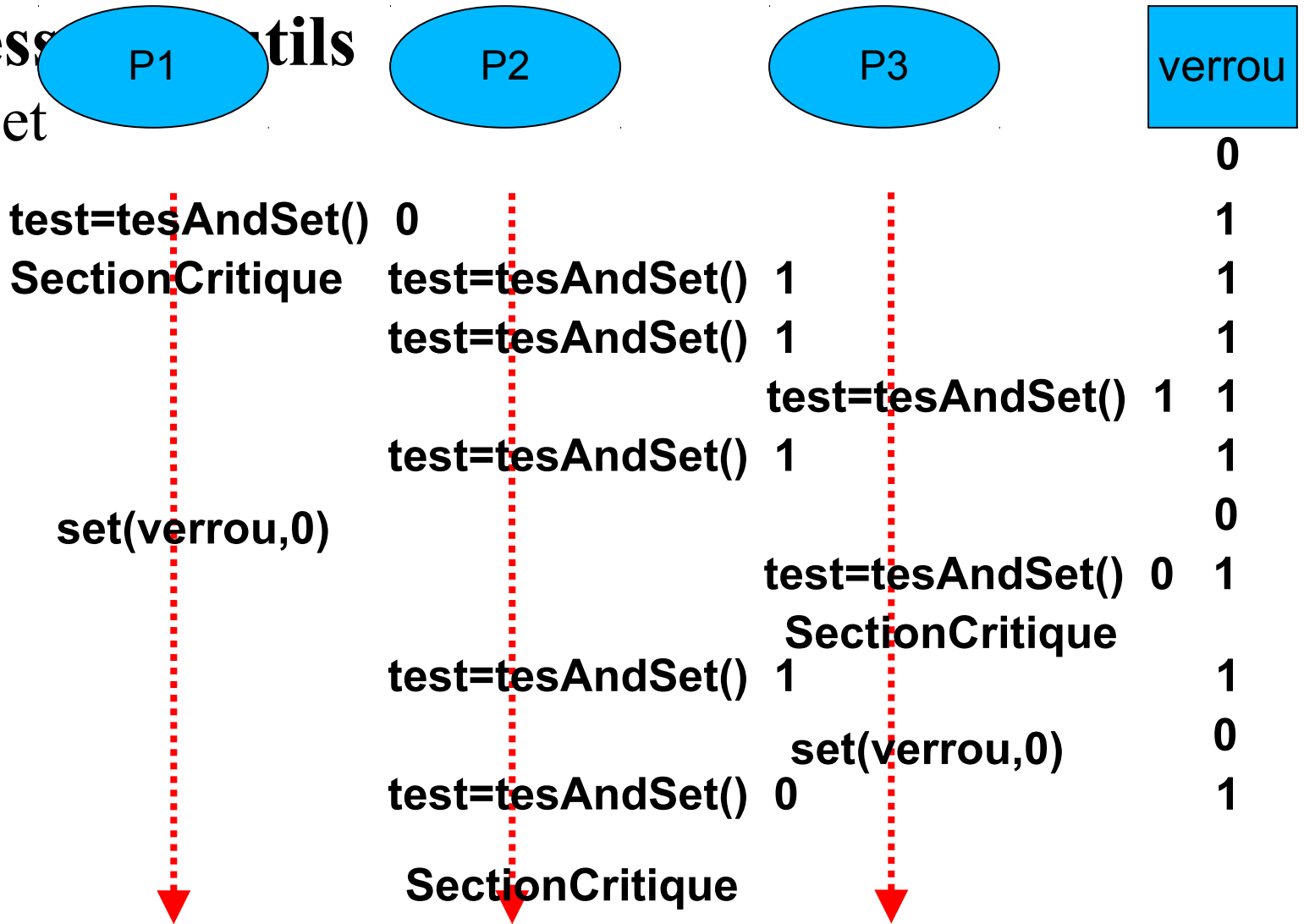
...

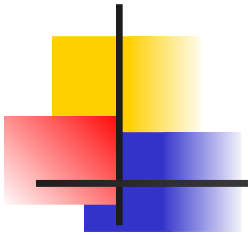
```
test = testAndSet(&verrou);
while (test  == 1) //le verrou est pris
  test = testAndSet(&verrou);
section_critique ...
set(&verrou, 0);
```

Concurrence/synchronisation

La synchronisation de processus utils

testAndSet





Concurrence/synchronisation

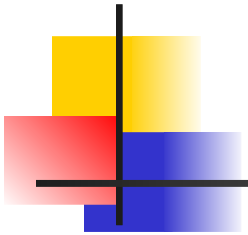
La synchronisation de processus :

testAndTest

inconvénients :

On ne garantit pas l'ordre. P2 a demandé la ressource avant P1 mais c 'est P1 qui l'obtient avant P2.

Si le verrou est pris, le processus boucle sur le testAndSet. On a donc une consommation de CPU. C'est de l' Attente Active



La synchronisation de processus : les sémaphores

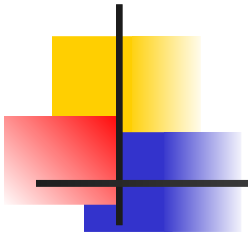
Les sémaphores Dijkstra 1965

Un sémaphore est une structure contenant un compteur et une file d ’attente.

La structure est manipulée par deux opérations

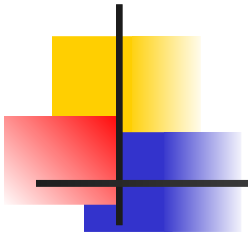
P et V, on ajoute souvent une opération supplémentaire permettant d ’initialiser la structure.

Ces trois opérations sont atomiques (non interruptibles et prises en charge par l’OS)



La synchronisation de processus : les sémaphores

```
typedef struct {
   int c;    // compteur de ressources
   file f;   // file d'attente de procesus
             // en attente de ressources
} semaphore;
```



La synchronisation de processus : les sémaphores

```
void Init(semaphore *s, unsigned v){
    s->c = v;
}
```

La valeur d'initialisation du sémaphore correspond au nombre d'exemplaires de ressources critiques.

Concurrence/synchronisation/sémaphores

La synchronisation de processus : les sémaphores

P Proberen (essayer)

```
void P(semaphore *s) {
   if (s->c  <= 0 )  {
        entrer(s->f,processus_courant)
        bloque(processus_courant)
   } else
        s->c --;
}
```

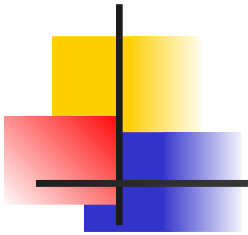
L'opération P demande une ressource et bloque le processus courant si il n'en reste plus .

Concurrence/synchronisation/sémaph

La synchronisation de processus : les sémaphores

```
V Vergohen (augmenter)
void V(semaphore *s) {
   if (nonVide(s->f))  {
        extraire(s->f,processus)
        etat_pret(processus)
   } else
        s->c ++;
}
```

L'opération V libère une ressource et débloque éventuellement un processus. La gestion des listes des processus en attente est FIFO. V débloque le plus vieux processus dans la file.



La synchronisation de processus : les sémaphores

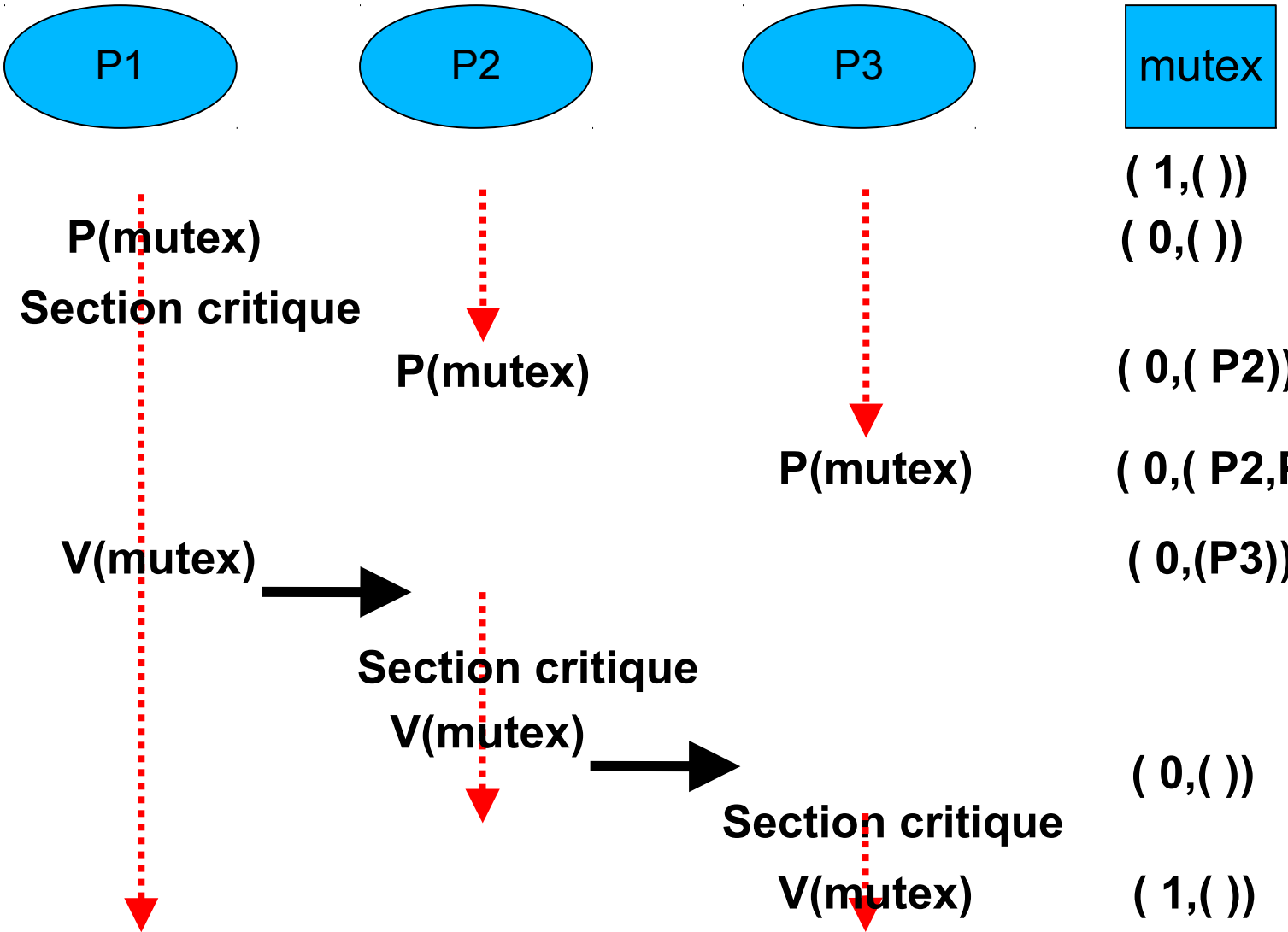
exemple : l'exclusion mutuelle

```
Init(&mutex,1);
```

…

```
P(&mutex)
section critique
V(&mutex)
```

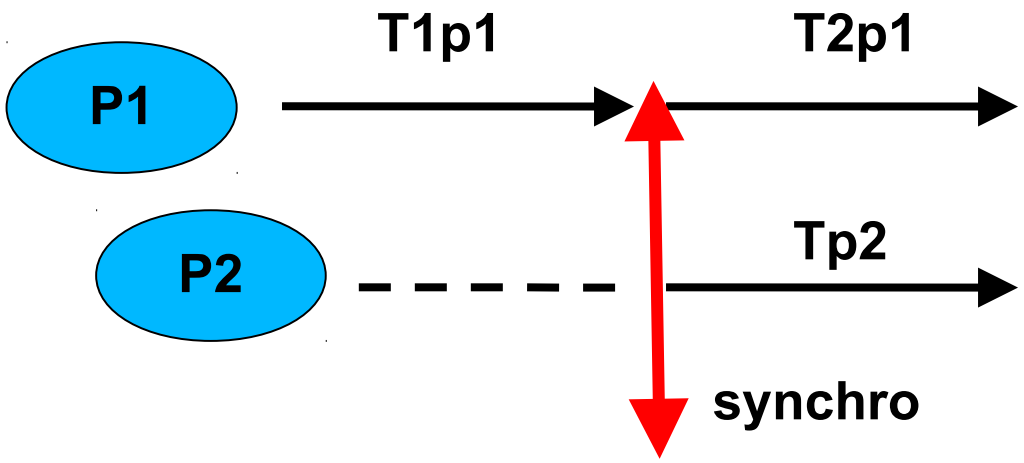
La synchronisation de processus : les sémaphores



La synchronisation de processus : les sémaphores

exemple : Synchro de 2 processus.

P1 effectue 2 tâches T1p1 et T2p1
P2 doit attendre la fin de T1p1 pour exécuter sa tâche Tp2



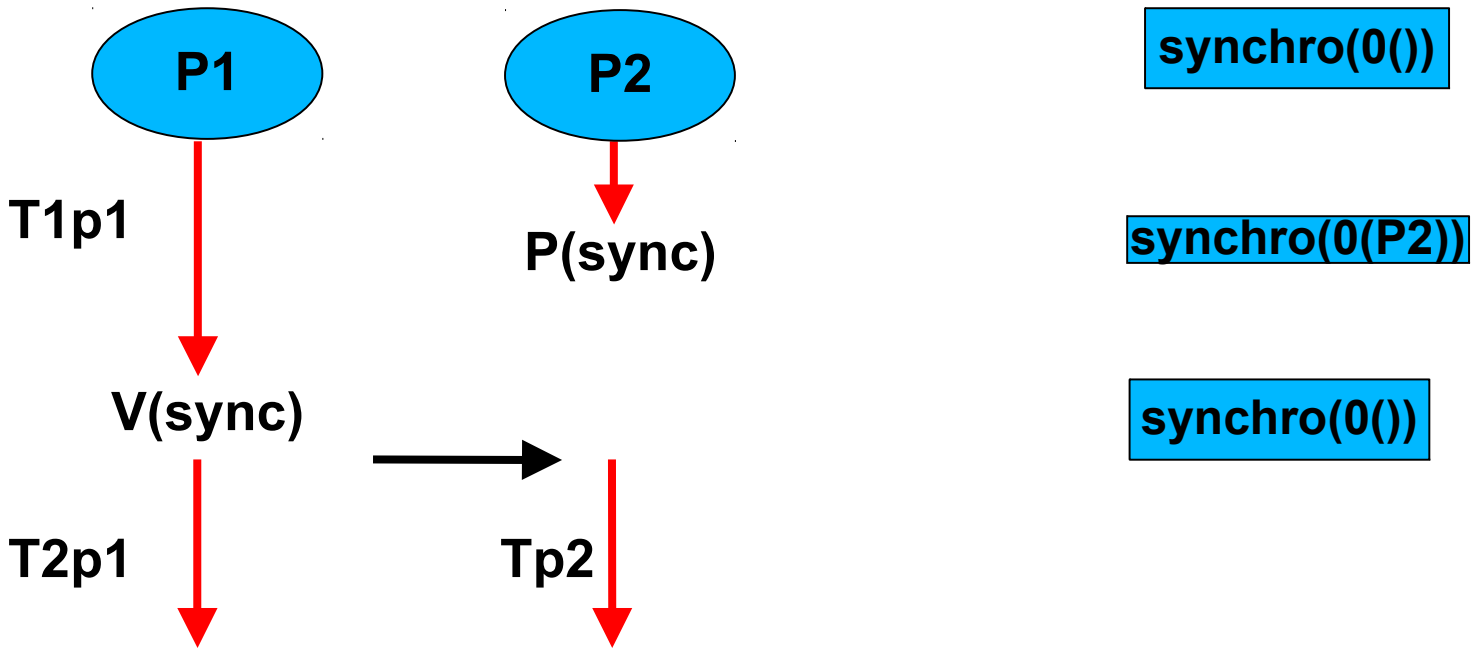
La synchronisation de processus : les sémaphores

exemple : Synchro de 2 processus.

```
Init(&synchro,0);
P2(void) {
      P(&sync); // attend la fin de T1p1
      Tp2();
}
P1(void) {
      T1p1(); // tache T1 de  P1
      V(&sync); // T1p1 est terminé
      T2p1():
}
```

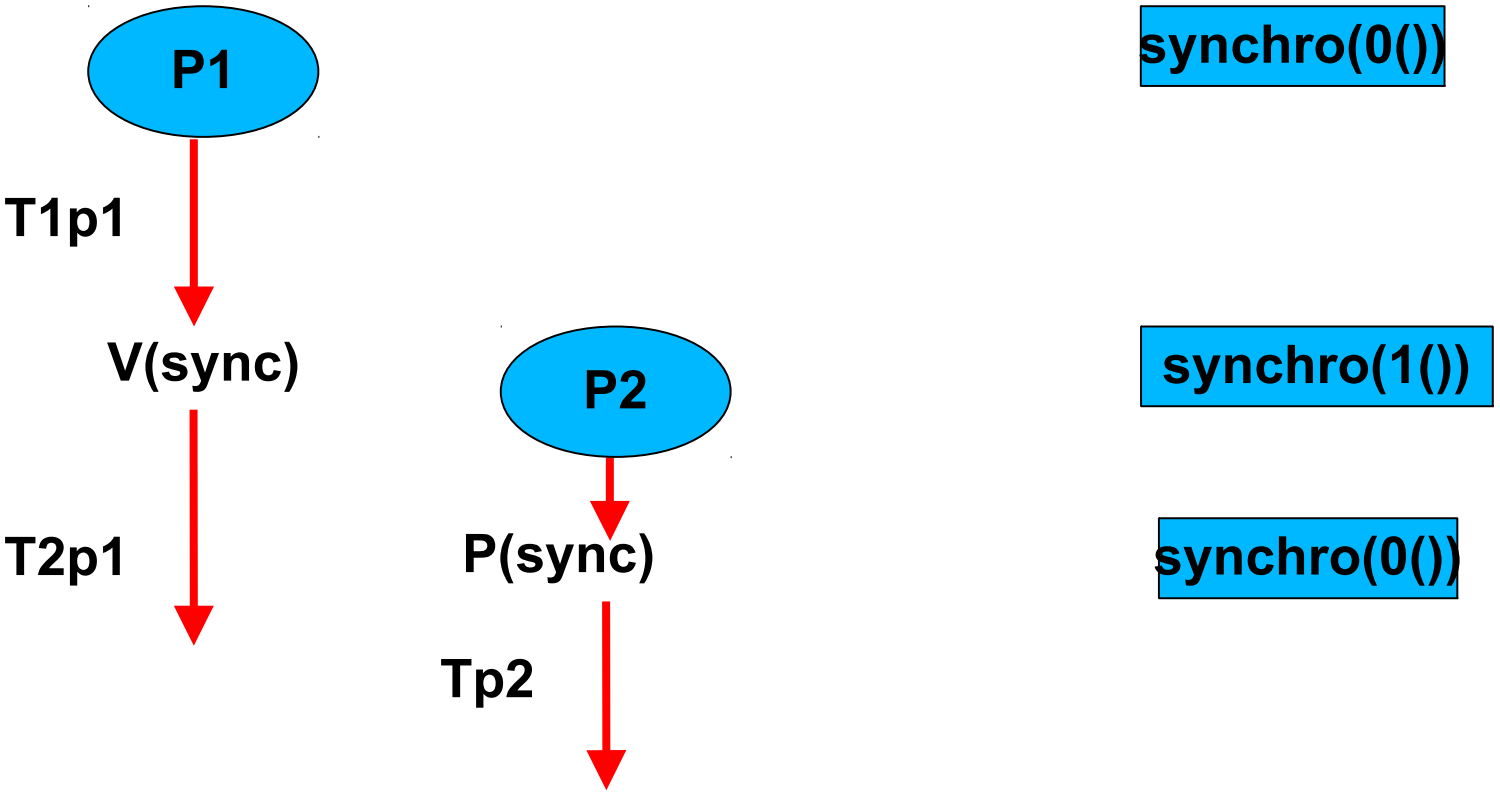
La synchronisation de processus : les sémaphores

exemple : Synchro de 2 processus.



La synchronisation de processus : les sémaphores

exemple : Synchro de 2 processus.



La synchronisation de processus : les sémaphores

Algorithme du Producteur-consommateur sur un tampon borné.

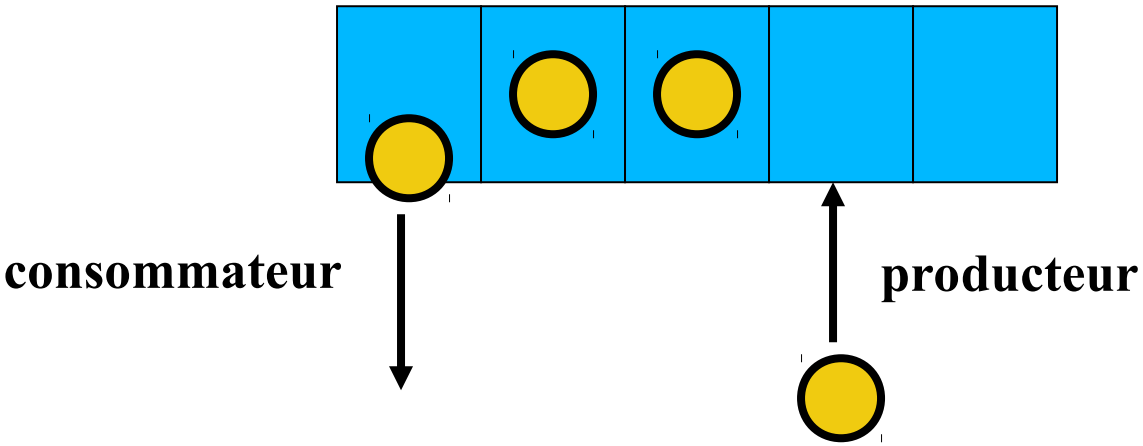
Deux processus se partagent une mémoire tampon de taille fixe (ex un tableau). Un des processus, le producteur, dépose des éléments dans le tampon. Le deuxième processus extrait l'information (le consommateur).

Les problèmes surviennent lorsque :

- le tampon est plein, le producteur ne peut plus déposer d'information.
- le tampon est vide, le consommateur ne peut plus extraire de l'information.

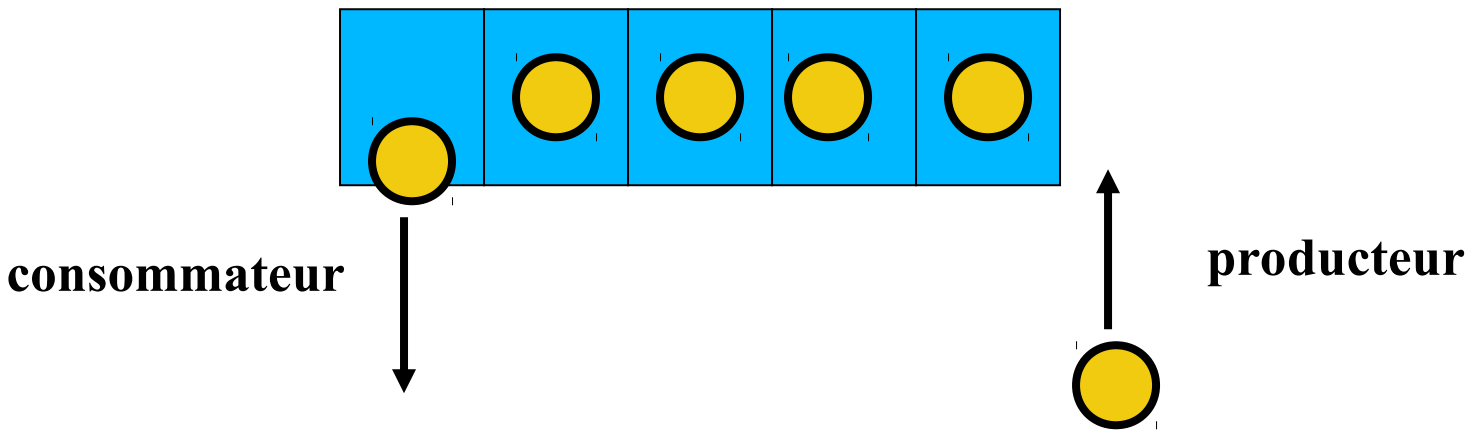
La synchronisation de processus : les sémaphores

Algorithme du Producteur-consommateur sur un tampon borné.



La synchronisation de processus : les sémaphores

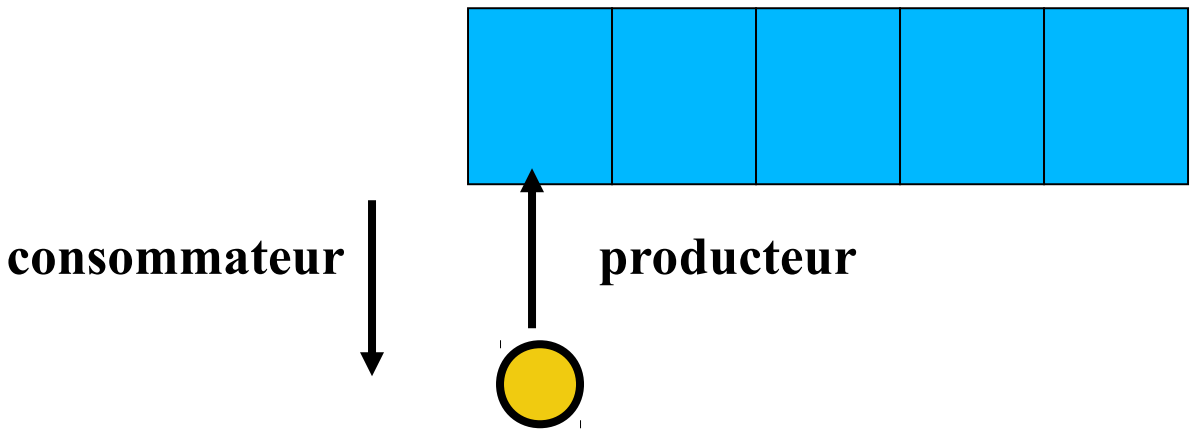
Algorithme du Producteur-consommateur sur un tampon borné.



Le tampon est plein, le processus producteur doit être bloqué jusqu'à ce qu'un consommateur retire un élément

La synchronisation de processus : les sémaphores

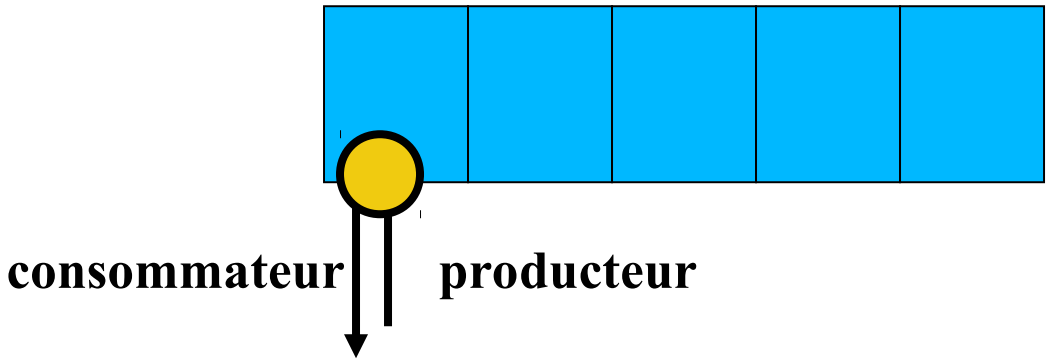
Algorithme du Producteur-consommateur sur un tampon borné.



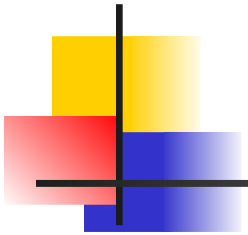
Le tampon est vide, le processus consommateur doit être bloqué jusqu'à ce qu'un producteur dépose un élément

La synchronisation de processus : les sémaphores

Algorithme du Producteur-consommateur sur un tampon borné.

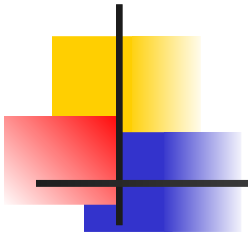


L ’accès aux cases du tampon doit être protégé. Le consommateur ne peut extraire l’élément que le producteur est entrain de déposer



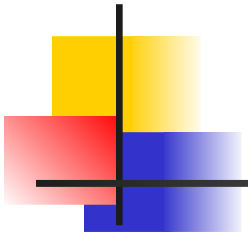
Producteur-consommateur sur un tampon borné.

```
#define MAX 100
   //taille du tampon
semaphore mutex;   // acces au tampon
semaphore places;
   // nombre de places disponibles dans le tampon
semaphore articles;
    // nombre d'articles dans le tampon
```



Producteur-consommateur sur un tampon borné.

```
void initialisation() {
   Init(& mutex,1);
   Init(& places,MAX);
   Init(& articles,0);
}
```



Concurrence/synchronisation/sémaphores

Producteur-consommateur sur un tampon borné.

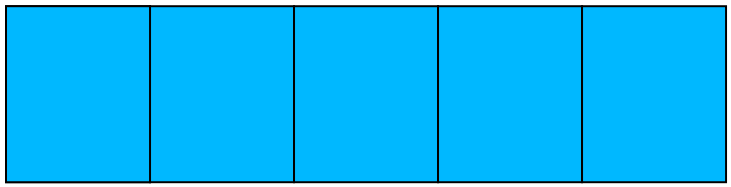
```
void producteur(void) {
 int objet;
 while (1) {
   produire(&objet);
   P(&places); // demande une place
   P(&mutex); //demande l'accès à la table
   deposer(objet);
   V(&mutex);
   V(&articles); // un article en plus
 }
}
```

Concurrence/synchronisation/sémaphores

Producteur-consommateur sur un tampon borné.

```
void consommateur(void) {
  int objet;
  while (1) {
    P(&articles); // demande un article
    P(&mutex);//demande l'accès a la table
    extraire(&objet);
    V(&mutex);
    V(&place)// une place ne plus
    consommer(objet);
  }
}
```

Producteur-consommateur sur un tampon borné.



consommateur

P(article)

extraire

producteur

produire

P(place)

deposer

V(article)

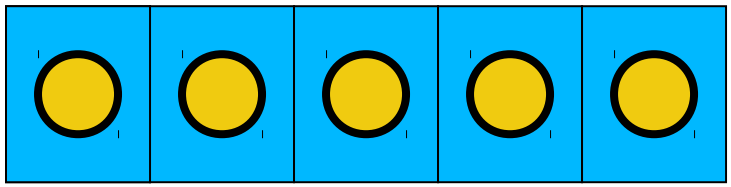
place ((5) ()) article ((0) ())

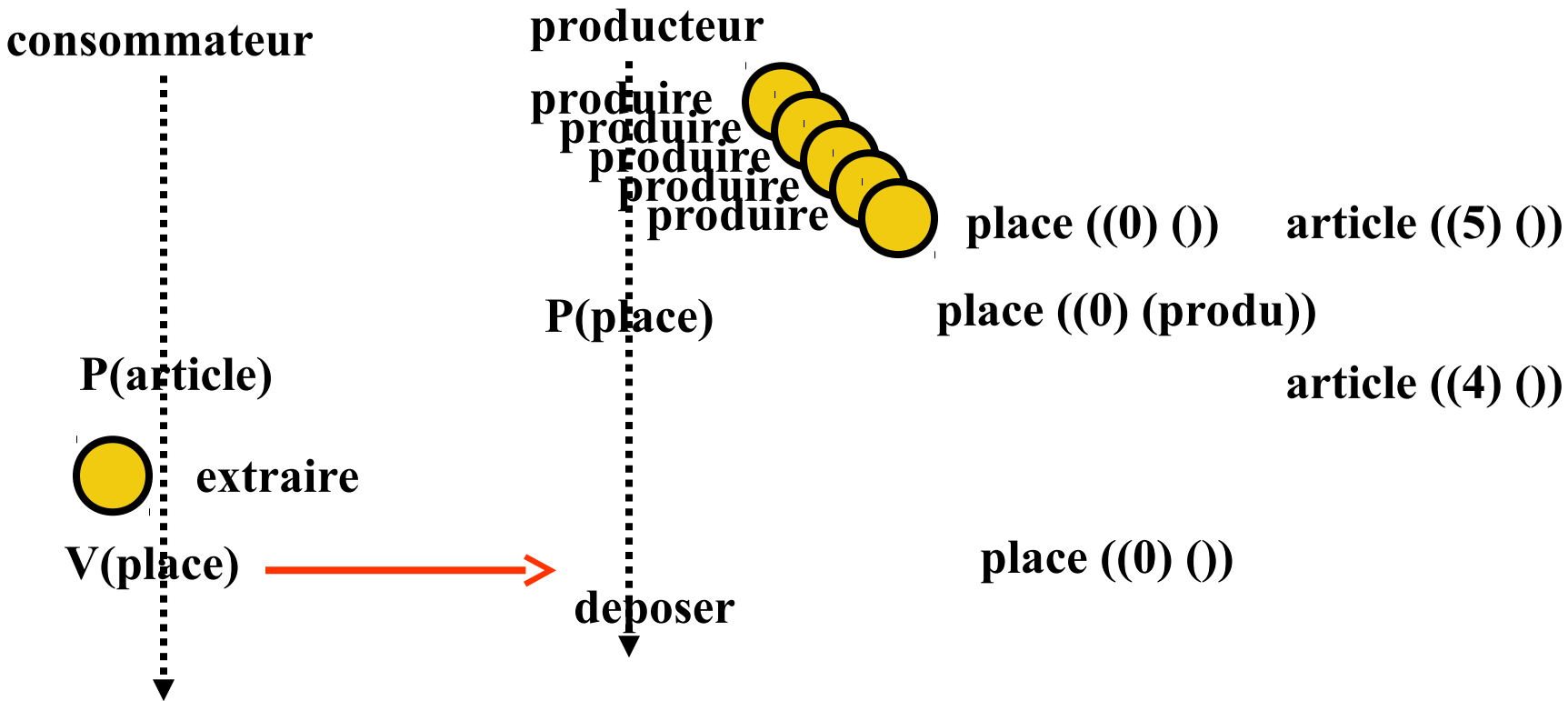
article ((0) (conso..))

place ((4) ())

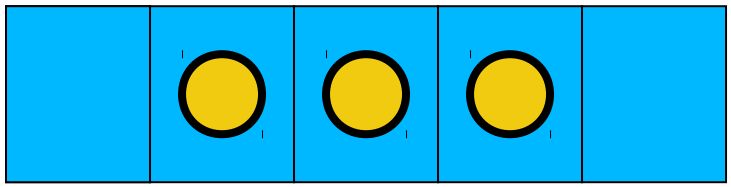
article ((0) ())

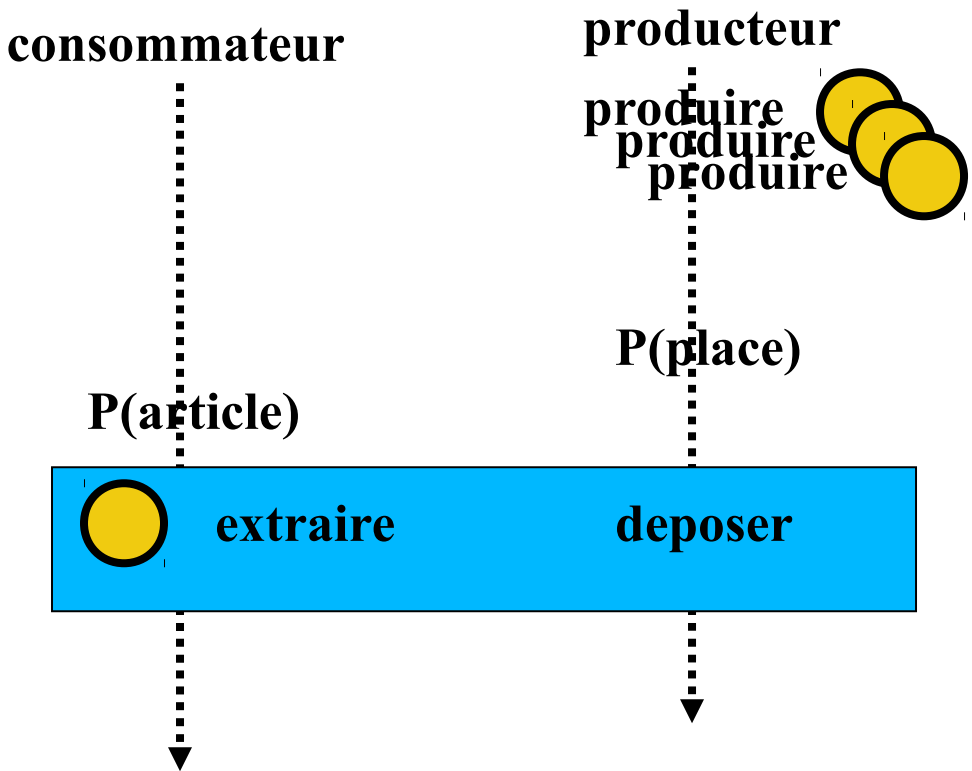
Producteur-consommateur sur un tampon borné.





Producteur-consommateur sur un tampon borné.



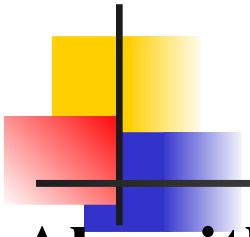


place ((2) ()) article ((3) ())

place ((1))

article ((4) ())

Le sémaphore mutex garantit que extraire et déposer sont séquentialisés
donc extraire puis déposer
ou déposer puis extraire



Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur :

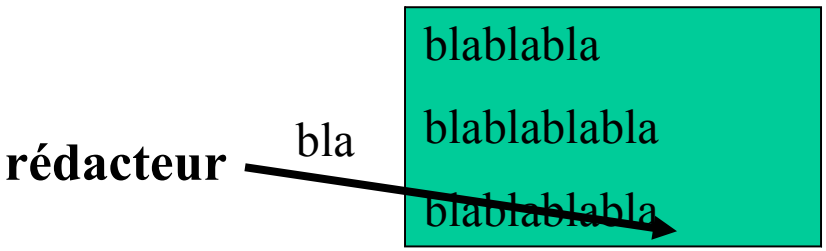
Les lecteurs peuvent consulter l'information qu'un rédacteur produit (rédige).

Les contraintes sont les suivantes :

- Il n'y a pas de limite sur le nombre de lecteurs en parallèle
- Les lectures s'effectuent en exclusion mutuelle avec le rédacteur
- Exclusion mutuelle entre les rédacteurs.

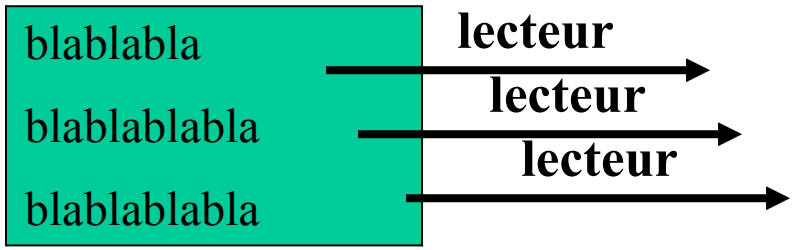
Algorithme de lecteur-rédacteur

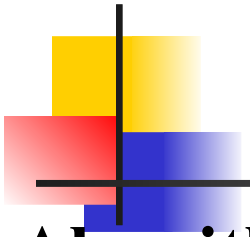
Le modèle lecteur(s)-rédacteur :



Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur :

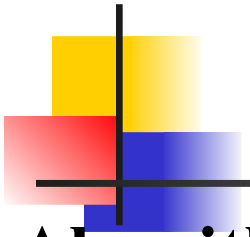




Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur :

```
semaphore info;          // acces à l'info
// variable partagée
"shared" int nbL;  // nombre de lecteurs
semaphore semNbL;   // protection de la variable
                       //  nbL nombre de lecteurs
void initialisation() {
  Init(& info,1);
  Init(& semNbL,1);
  nbL=0);
}
```



Algorithme de lecteur-rédacteur

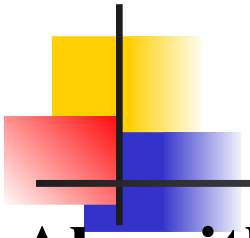
Le modèle lecteur(s)-rédacteur :

```
void lecteur(void) {
     P(&semNbL); // demande l'accès à nbL
     nbL++;
     if (nbL ==1) // seul lecteur
        P(&info); // demande l'accès à l'info
                  // en exclusion avec le rédac.
     V(&semNbL); // libère nbL
     ....
     accès en lecture à l'information
```

Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur :

```
        accès en lecture à l'information
        ...
        P(&semNbL); // demande l'accès à nbL
        nbL--;
        if (nbL ==0) // seul lecteur
              V(&info);// libère l'accès à l'info
                       // éventuellement le rédac
        V(&semNbL); // libère nbL
}
```

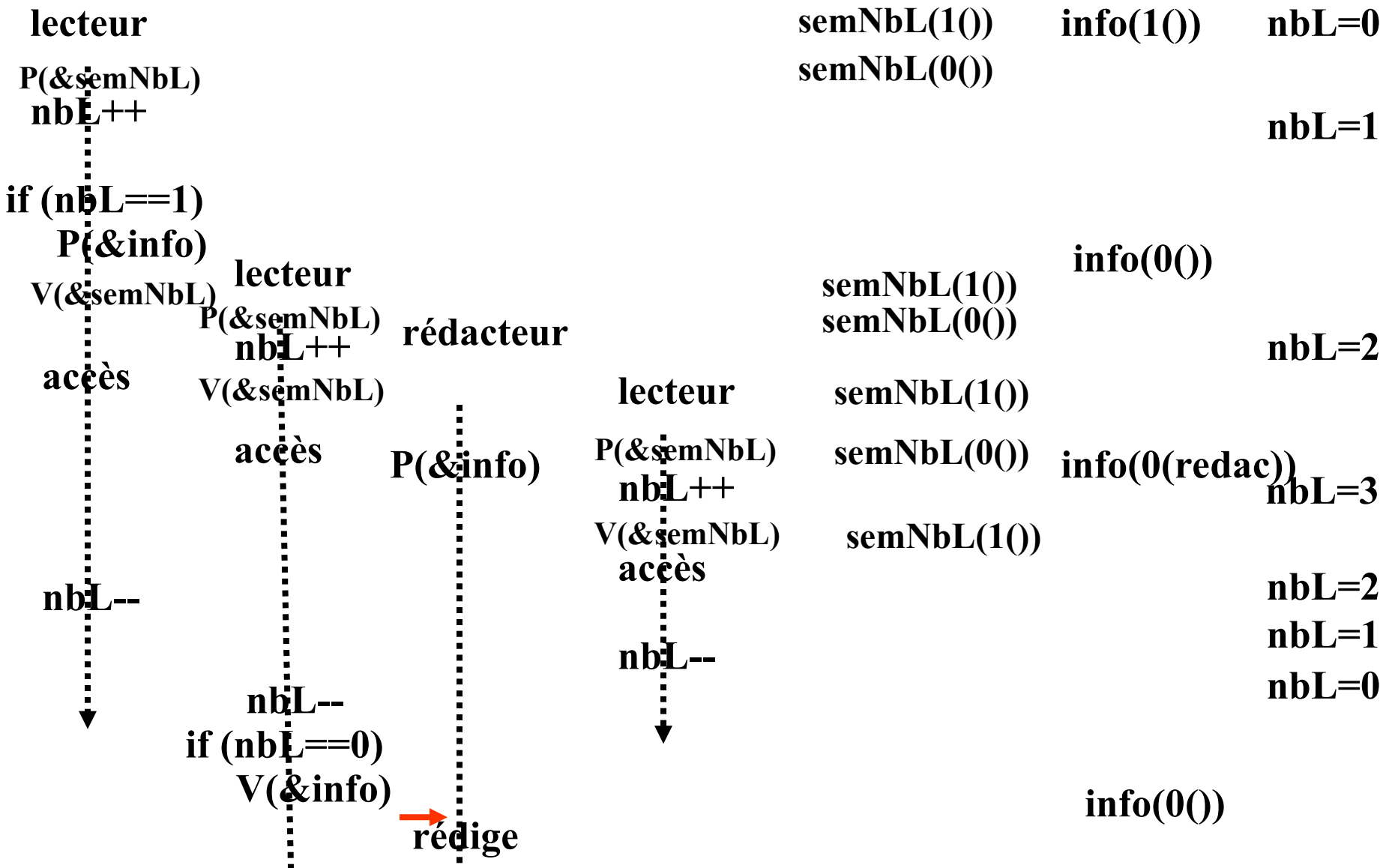


Algorithme de lecteur-rédacteur

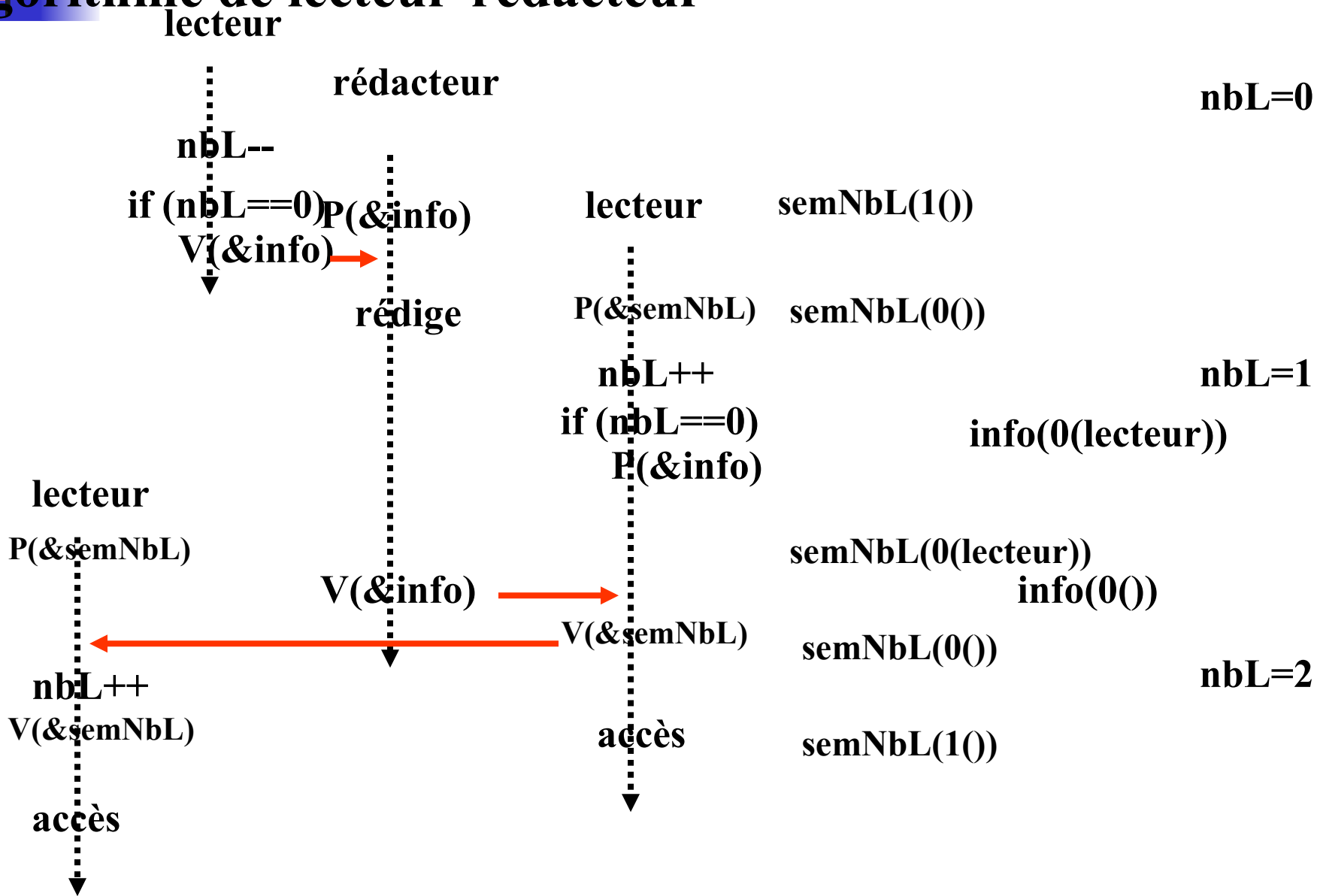
Le modèle lecteur(s)-rédacteur :

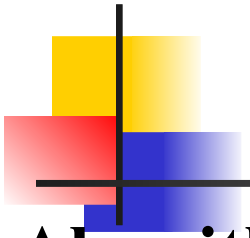
```
void redacteur(void) {

      P(&info);// demande l'accès à l 'info
        ... rédige l'information
      V(&info);

}
```

Algorithme de lecteur-rédacteur



Algorithme de lecteur-rédacteur

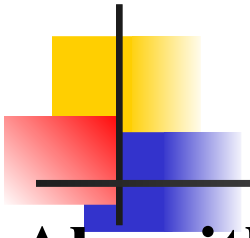




Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur :

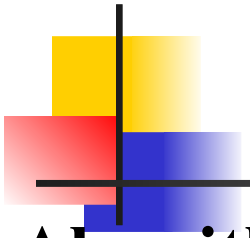
Ici il y a une priorité des lecteurs sur le rédacteur. Puisque le rédacteur doit attendre qu'il n'y ait plus de lecteurs en cours. Et si il y a déjà un lecteur, les nouveaux lecteurs passent avant le rédacteur.



Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur avec priorité du rédacteur

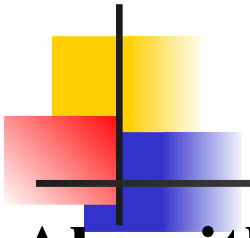
```
static    int lecteur,demandeLecteur;
          // nombre de lecteurs
static    int redacteur,demandeRedacteur;
semaphore mutex;  // protection des variables
semaphore semLec;
semaphore semRed;
```



Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur avec priorité du rédacteur

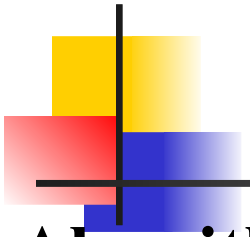
```
void initialisation() {
   Init(& mutex,1);
   Init(& semLec,0);
   Init(& semRed,0);
   lecteur=0;
   demandeLecteur=0;
   redacteur=0;
   demandeRedacteur =0;
}
```



Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur avec priorité du rédacteur

```
void lecteur(void) {
    P(&mutex); // accès aux variables protégées
    if (redacteur || demandeRedacteur)
       // test il si y a un rédateur ou
       // une demande d'un rédacteur
        demandeLecteur++;
        V(&mutex)
        P(&semLec);
        P(&mutex)
        demandeLecteur--;
    lecteur++;
    V(&mutex);
```



Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur avec priorité du rédacteur

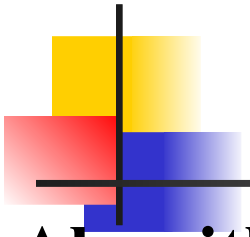
….

```
 accès en lecture à l'information
      // sortie ...
      P(&mutex);
      lecteur--;
      if (lecteur == 0 &&  demandeRedacteur)
            // seul lecteur
             V(&semRed);// libere le rédacteur
      V(&mutex);
}
```

Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur avec priorité du rédacteur

```
void redacteur(void) {
    P(&mutex); // demande au variables
    if (lecteur||redacteur || demandeRedacteur){
          demandeRedacteur++
          V(&mutex)
          P(&semRed);
          P(&mutex)
          demandeRedacteur --;
    }
    redacteur ++;
    V(&mutex)
     ...
     // rédige l'information
```



Algorithme de lecteur-rédacteur

Le modèle lecteur(s)-rédacteur avec priorité du rédacteur

```
    ...
      P(&mutex)
      redacteur --;
      if (demandeRedacteur)
            V(&semRed)
      else {
         if (demandeLecteur) {
           int nb;
           for (nb=0;nb < demandeLecteur, nb++)
             V(&semLec);
         }
      }
      V(&mutex);
}
```

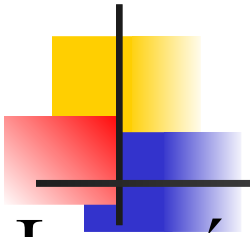
le problème du "dead lock"}

```
semaphore s1;
semaphore s2;

void initialisation() {
   Init(& s1,1);
   Init(& s2,1);
}
```

Le problème du "dead lock"

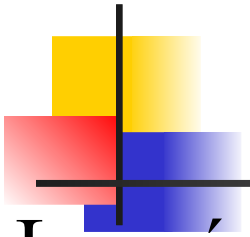
```
void A(void) {
      ...
      P(&s1);
      P(&s2);
      ...
}
void B(void) {
      ...
      P(&s2);
      P(&s1);
      ...
}
```

Ici on peut se retrouver avec une situation d 'inter-blocage.



Les sémaphores IPC

Les IPC (Inter Proccess Communication) sont souvent ajoutées au noyau du système Linux. Ils proposent un ensemble de services : sémaphores, segments partagés et une bibliothèque de messages entre processus d'une même machine.



Les sémaphores IPC

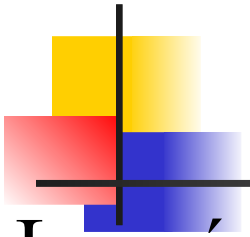
#include <sys/sem.h>

int semget(key_t nom_ext,int nb, int acces)

Création d'un ensemble de nb sémaphores

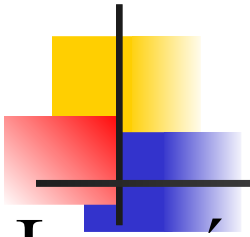
Le premier appel à la primitive crée un ensemble de nb sémaphores. Les droits d'accès (int acces) sont(IPC_CREAT | 0666) ou NULL si l'ensemble existe déjà. La fonction retourne un nom interne de l'ensemble. IPC_PRIVATE dans acces crée des sémaphores privés au processus et à ses fils.

Aux appels suivants avec la même clé la fonction retourne un nom interne de l'ensemble des sémaphores.



Les sémaphores IPC

```
struct sembuf {
  unsigned short int sem_num;
        // numero de semaphore
        //  0 est le premier de l'ensemble
  short sem_op;
        // numero d'operation
  short sem_flg;
        // option
}
```



Les sémaphores IPC

Définition d'une opération

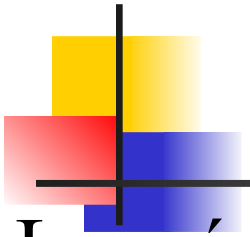
sem_num : numéro de sémaphore 0 à n-1

sem_op : type de l'opération

1 -> V(s) (ou plus n,incrémente de n le compteur)

-1 -> P(s) (ou plus n,décrémente de n le compteur (appel bloquant))

0 -> Z(s) bloqué jusqu'à ce que la valeur du compteur soit égale à 0

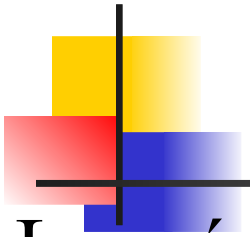


Les sémaphores IPC

Définition d'une opération

sem_flg : option de l'opération :

- **NULL par défaut**
- **IPC_NOWAIT opération ne sera pas effectuée si elle est bloquante (vérifier errno a EAGAIN)**
- **SEM_UNDO (operation inverse en fin de processus)**

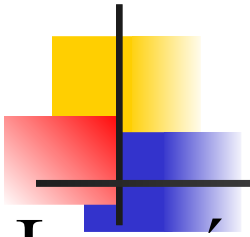


Les sémaphores IPC

```
int semop(int nom,
          struct sembuf *tab_op,
          int nb_op)
```

La fonction permet d'effectuer une liste d'opérations sur un ensemble de sémaphores.

- **nom : nom interne de l'ensemble**
- **nb_op : nombre d'opérations à effectuer (taille du tableau)**
- **tab_op : tableau des opérations**

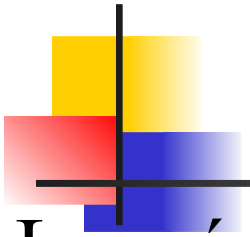


Les sémaphores IPC

```
int semctl(int nom, int semnum, int op,
  union senum ARG);
```

contrôle un ensemble : lire les valeurs, les positionner ou les initialise, détruire l'ensemble des sémaphores, ...

```
union senum{
   int val;
   struct semid * buf;
   ushort * array;
}
```



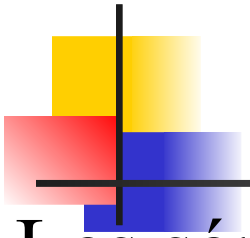
Les sémaphores IPC

nom : nom interne de l'ensemble

semnum : numéro ou nombre de sémaphores

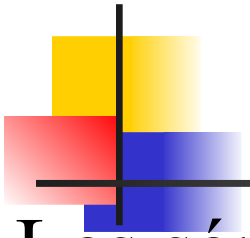
op : type de l'opération
SETVAL,SETALL,GETVAL,GETALL,IPC_RMID

ARG : options de l'opération par exemple le tableau des valeurs d'initialisation des sémaphores



Les sémaphores IPC

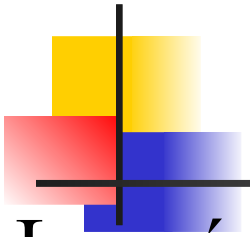
```
> ipcs
ipc status (sémaphores, messages et
  segments partagés).
------ Semaphore Arrays --------
key        semid      owner      perms      nsems

0x00280269 128        dupont       666        14

> ipcrm sem 128
```



Concurrence/synchronisation/ sémaphores

Les sémaphores IPC

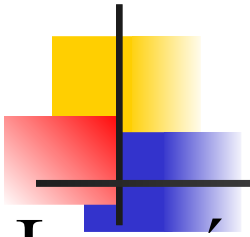
```
>  ipcs -s -i 128
Tableaux de sémaphores semid=622594
uid=261   gid=200     cuid=261       cgid=200
mode=0666, access_perms=0666
nsems = 1
otime = Sat Sep 22 11:37:57
ctime = Sat Sep 22 11:37:57
semnum    valeur    ncount    zcount      pid

0           1           0         0         3384
```



Les sémaphores IPC

exemple: partage de la sortie standard.

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

key_t cle; // cle ipc
int semid;//nom local de l'ens. des sémaphores
```



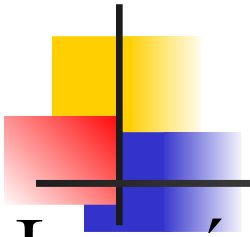
Concurrence/synchronisation/ sémaphores

Les sémaphores IPC :partage de la sortie standard.

```
void out(char *s) {
    struct sembuf op;
    //P(&mutex);
    op.sem_num=0;op.sem_op=-1;op.sem_flg=0;
    semop(semid,&op,1);
    //  en exclusion mutuelle
    write (1,s,strlen(s))
    //V(&mutex);
    op.sem_num=0;op.sem_op=1;op.sem_flg=0;
    semop(semid,&op,1);
 }
```

Les sémaphores IPC :partage de la sortie standard.

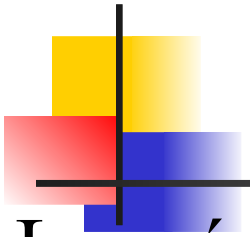
```
int main ( int argc , char **argv ) {

  //file 2 key
  // construction de la clé
  // à partir d'une  entrée unix
  // key_t (char *pathname,char proj)
  // inode & 0xFFFF| st_dev<<16| proj << 24)
  if ((cle=ftok(argv[0],'0')) == -1 ) {
    fprintf(stderr,"Problème sur ftok\n");
    exit(1);
  }
```



Concurrence/synchronisation/ sémaphores

Les sémaphores IPC :partage de la sortie standard.

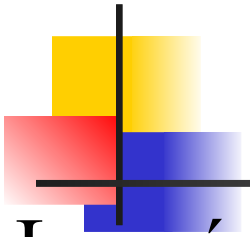
```
...

if ((semid=semget(cle,1/*un semaphore*/,
                  IPC_CREAT|IPC_EXCL|0666))==-1) {
                 // IPC_EXCL retourne erreur
                 // si l 'ensemble existe deja
  fprintf(stderr,"Probleme sur semget\n");
  exit(2);
}
```



Concurrence/synchronisation/ sémaphores

Les sémaphores IPC :partage de la sortie standard.

```
{
  ushort init_sem[1]={1};
  if (semctl(semid,1,SETALL,init_sem)==-1) {
    // 1 nombre de sémaphores à initialiser
   fprintf(stderr,"Probleme sur semctl \n
                          SETALL\n");
   exit(3);
 }
}
```



Concurrence/synchronisation/ sémaphores

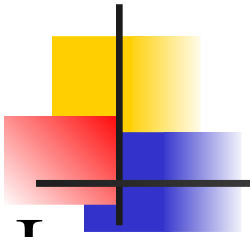
Les sémaphores IPC :partage de la sortie standard.

```
{ // création des taches
 pid_t pid;int i;
 for (i=1 ;i < argc; i++) {
     if ((pid =fork())==0) {
       out(argv[i]);
       exit(0);
     }
 }
 // attente de la fin des fils
 for (i=1 ;i < argc; i++) wait(NULL);
}
```

Concurrence/synchronisation/signaux

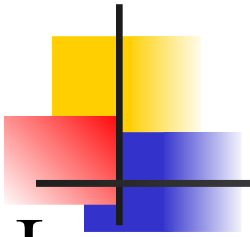
- Problème de synchronisation
- TestEtSet
- Sémaphores
- Monitors
- Segments partagés (mémoire)
- Signaux
-

Système d'exploitation : concurrence



Les moniteurs.

- **Hoare 74**
- **un moniteur :**
- **Abstraction de programmation (pour le développeur)**
  - **Ensemble de variables (données)**
  - **Ensemble de procédures (code)**
  - **Accès au moniteur en exclusion mutuelle (OS)**
  - **Des conditions avec deux opérations wait et signal(ou notify)**

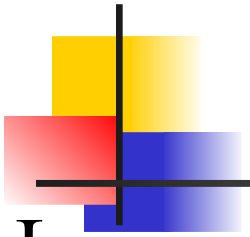


Les moniteurs.

opérations : wait et signal (en exclusion mutuelle)

Les conditions permettent la mise en attente des processus

- **wait : le processus courant attend sur une condition.**
- **Signal : réveille tous les processus en attend sur le signal (en fin de procedure)**

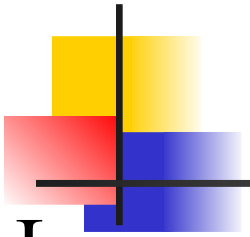


Les moniteurs.

exemple du producteur consommateur

(en pseudo C++)

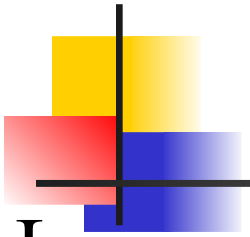
```
#define MAX 100
Class producteur-consommateur : public moniteur {
  protected
     condition plein, vide;
     int compteur;
  public :

   producteur-consommateur(void){
     compteur = 0;
  }
```



Les moniteurs.

exemple du producteur (en pseudo C++)

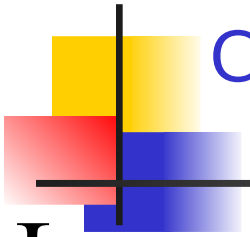
```
// procedure  (toujours en exclusion mutuelle)
void deposer(void) {
  while (compteur == MAX)
    wait(plein); //attente
                 // (perd l'exclusion mutuelle)
  //recupere exclusion mutuelle
  .. Deposer objet
  compteur ++;
  if (compteur  == 1)
       signal(vide);
} // libère l'exclusion mutuelle
```



Les moniteurs.

exemple du consommateur (en pseudo C++)

```
// procedure  (toujours en exclusion mutuelle)
void extraire(void) {
  while (compteur == 0)
    wait(vide); //attente
  //recupere exclusion mutuelle
  .. Extrait objet
  compteur --;
  if (compteur  == MAX -1 )
        signal(plein);
} // libère l'exclusion mutuelle
```

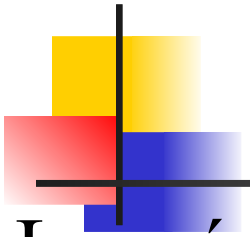


La mémoire partagée

Les segments partagés IPC.

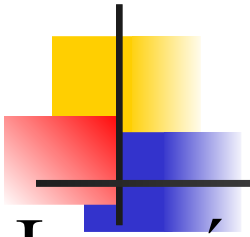
Les processus UNIX ont leurs propres segments de données. Les IPC permettent à plusieurs processus de se partager de la mémoire en lecture/ecriture (un segment de données partagé).

Les processus doivent alors se synchroniser sur les accès à cette mémoire (sémaphores).



La mémoire partagée IPC.

```
#include <sys/shm.h>
int shmget(key_t nom_ext,int size, int acces)
Création d'un segment partagé
Le premier appel à la primitive crée un segment
  partagé de  size octets. Taille doit être un
  multiple de PAGE_SIZE.
 Les droits d'accès sont(IPC_CREAT | 0666) ou
  NULL si le segment existe déjà. La fonction
  retourne un nom interne.
 Champs clé IPC_PRIVATE crée un segment partagé
  privé au processus et ses fils.
  Aux appels suivants avec la même clé la
  fonction retourne le même nom interne du
  segment.
Vérifier errno (ENOMEM mémoire insuffisante)
```

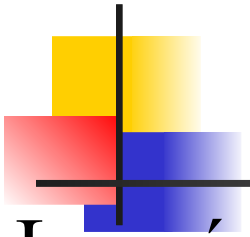


La mémoire partagée IPC.

void *shmat(int shmid,const char *adr, int flags)

Attache le segment partagé dans la zone adressable du processus. Si adr vaut NULL, le système retourne une adresse qu'il choisit sinon il retourne l'adresse fixée par le programmeur ou une adresse la plus proche.

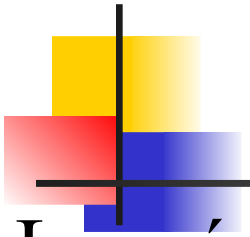
Si flags vaut SHM_RDONLY, le processus n'accédera au segment qu'en lecture (si flags vaut NULL par défaut lecture-écriture).



La mémoire partagée IPC.

```
int shmdt( const char *adr )
```

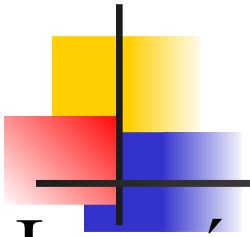
détache le segment partagé attaché à l'adresse adr de la zone adressable du processus.



La mémoire partagée IPC.

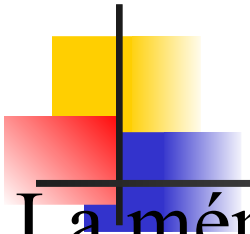
```
int shmctl(int nom,int cmd,struct shmid_ds*buf);
contrôle le segment :
avec cmd :
  IPC_STAT récupère le status du segment
  IPC_SET modifie le status du segment
        (champs uid,gid, mode)
  IPC_RMID détruit le segment

avec la structure shmid_ds suivante:
```

Concurrence/synchronisation/segment partagé



La mémoire partagée IPC.

```
struct shmid_ds{
  struct ipc_perm shm_perm // key
                           // uid gid
                           // mode  0666
  int shm_segsz;  // taille en octets du seg
  time_t shm_atime; // date du dernier shmat
  time_t shm_dtime; // date du dernier shmdt
  unsigned short shm_cpid; // pid créateur
  unsigned short shm_lpid; // pid last opération
  short  shm_nattch;  // nombre de shmat actuel
  unsigned long * shm_pages; //table de pages
  ..
}
```

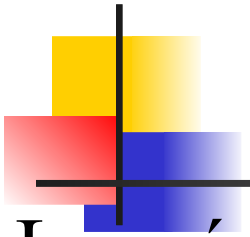


La mémoire partagée IPC.

Un processus met une donnée à disposition d'autre processus

Code du producteur avant les autres processus

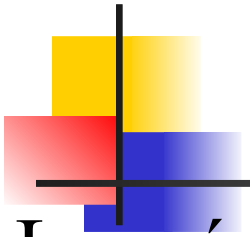
```
#include <sys/ipc.h> // IPC
#include <sys/shm.h> // shared memory
int main (   ) {
   key_t cle; int shmid;
   if ((cle=ftok(/etc/mdj,'0')) == -1 ) {
     fprintf(stderr,"Problème sur ftoks\n");
     exit(1);
   }
   if ((shmid=shmget(cle,4096,
                  IPC_CREAT|IPC_EXCL|0644))==-1) {
     fprintf(stderr,"Probleme sur shmget\n");
     exit(2);
```



Concurrence/synchronisation/segment partagé

La mémoire partagée IPC.

```
 ..
{
 char * str;
 if ((str=(char *)shmat(shmid,NULL,NULL))==-1) {

     fprintf(stderr,"Probleme sur shmat\n");
     exit(2);
   }
 // met l'information
 strcpy(str, "il fait beau aujourd'hui");
 shmdt(str);
}
}// main  producteur
```



La mémoire partagée IPC.

Les processus lecteurs sans synchronisation

```
int main (  ) {
  key_t cle; int shmid;
  // construit la même clé
  if ((cle=ftok(''/etc/mdj'','0')) == -1 ) {
    fprintf(stderr,"Problème sur ftoks\n");
    exit(1);
  }
  // recupère le segment partagé
  if ((shmid=shmget(cle,4096,NULL))==-1) {

    fprintf(stderr,"Probleme sur shmget\n");
    exit(2);
  }
```

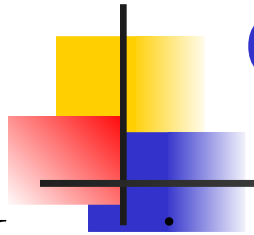
Concurrence/synchronisation/segment partagé

La mémoire partagée IPC.

```
{
char * str;
if ((str=shmat(shmid,NULL,SHM_RONLY))==-1) {

     fprintf(stderr,"Probleme sur shmat\n");
     exit(2);
   }
// affiche l'information
printf(str);
shmdt(str);
}
}// main lecteur
//ATTENTION IL FAUT QUE LE PRODUCTEUR AIT TERMINE
```

Concurrence/synchronisation/signaux

- Problème de synchronisation
- TestEtSet
- Sémaphores
- Monitors
- Segments partagés (mémoire)
- Signaux
-



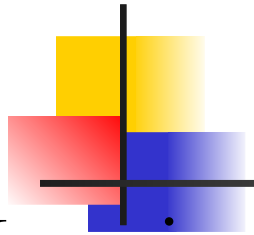
Concurrence/synchronisation/signaux

Les signaux norme POSIX

Un signal est une synchronisation asynchrone.

Un signal est engendré par un événement, un autre processus, le noyau. Il est envoyé à un ou plusieurs processus.

Le processus peut s'arrêter pour traiter le signal ou l'ignorer .



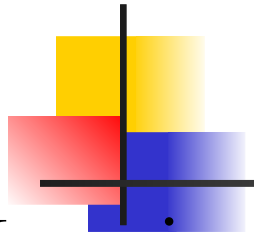
Concurrence/synchronisation/ signaux

Les signaux norme POSIX

Émission d'un signal

- Exception matérielle (IO ..)
- Terminal (Ctrl C Ctrl \ )
- Appel système kill(num) ou la commande kill -num
- Processus lui-même SIGSEGV SIGFPE
- Noyau (alarme)

NB : Certains signaux ne peuvent pas être interceptés KILL, STOP. CONT ...

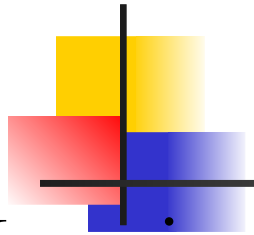


Les signaux norme POSIX

Liste des signaux signal.h (/usr/include/bits/signum.h)

```
SIGHUP      1     /* deconnexion d'un terminal */
SIGINT      2     /*  CTRL C
SIGQUIT     3     /* (*) quit, CTRL \ */
SIGILL      4     /* (*) illegal instruction (not
  reset when caught)*/
SIGTRAP     5     /* (*) trace trap (not reset when
  caught) */
SIGABRT     6     /* (*) abort process */
SIGEMT      7     /* EMT intruction */
SIGFPE      8     /* (*) floating point exception
```

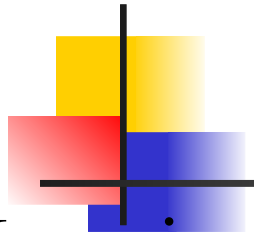
Concurrence/synchronisation/ signaux



Les signaux norme POSIX

```
SIGKILL     9     /* kill (cannot be caught or
  ignored) kill -9 */
SIGBUS     10     /* bus error (specification
  exception) */
SIGSEGV    11     /* segmentation violation */
SIGSYS     12     /* bad argument to system call */
SIGPIPE    13     /* write on a pipe with no one to
  read it */
SIGALRM    14     /* alarm clock timeout */
SIGTERM    15     /* software termination signal
  kill par défaut*/
SIGURG     16     /* (+) urgent contition on I/O
  channel */
```

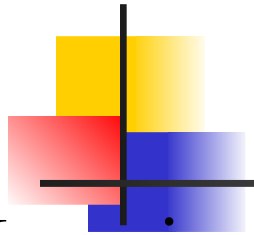
Concurrence/synchronisation/ signaux



Les signaux norme POSIX

```
SIGURG    16    /* (+) urgent contition on I/O
  channel */
SIGSTOP   17    /* (@) stop (cannot be caught or
  ignored) */
SIGTSTP   18    /* (@) CTRL Z */
SIGCONT   19    /* (!) continue (fg, bg ..) */
SIGCHLD   20    /* (+) sent to parent on child
  stop or exit */
SIGTTIN   21    /* (@) background read attempted
  from control terminal*/
SIGTTOU   22    /* (@) background write attempted
  to ccontrol terminal*/
```

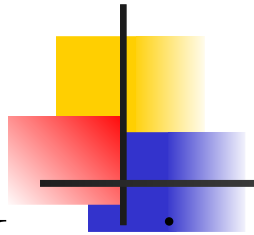
Concurrence/synchronisation/ signaux



Les signaux norme POSIX

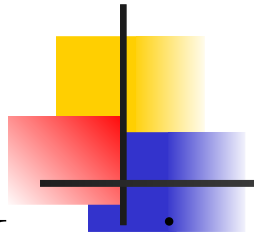
```
SIGIO      23    /*(+)I/O possible, or completed*/
SIGXCPU    24    /* cpu time limit exceeded (see
  setrlimit)*/
SIGXFSZ    25    /* file size limit exceeded (see
  setrlimit)*/
SIGMSG     27    /* input data is in the HFT ring
  buffer */
SIGWINCH   28    /* (+) window size changed */
SIGPWR     29    /* (+) power-fail restart */
```

Concurrence/synchronisation/ signaux



Les signaux norme POSIX

```
SIGUSR1    30     /* user defined signal 1 */
SIGUSR2    31     /* user defined signal 2 */
SIGPROF    32     /* profiling time alarm (see
  setitimer$
SIGDANGER 33      /* system crash imminent; free up
  some$
SIGVTALRM 34      /* virtual time alarm (see
  setitimer) $
SIGMIGRATE 35     /* migrate process (see TCF)*/
SIGPRE     36     /* programming exception */
```



Concurrence/synchronisation/ signaux

Les signaux norme POSIX

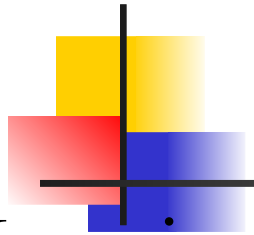
l 'appel système kill

```
int kill(pid_t pid, int sig);
```

envoie le signal numéro sig

Avec pid :

- pid > 0 au processus pid
- pid == 0 aux processus du groupe courant (celui qui effectue le kill)
- pid == -1 à tous les processus sauf init (0)
- pid < 0 aux processus du groupe |pid|



Concurrence/synchronisation/ signaux

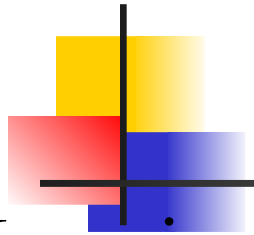
Les signaux norme POSIX

La commande UNIX kill

> kill -9 pid

```
SIGKILL     9     /* kill (cannot be caught or
  ignored)   */
```

> kill -KILL pid



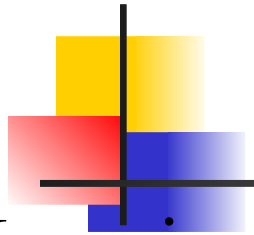
Concurrence/synchronisation/ signaux

Les signaux norme POSIX

l'appel système raise

```
int raise(int sig)
```

envoie le signal numéro sig au processus courant



Concurrence/synchronisation/ signaux

Les signaux norme POSIX

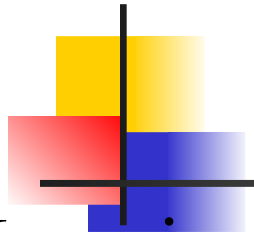
Action à effectuer à la réception d'un signal

Etape 1: Définir une fonction C qui sera appelée à la réception d 'un signal.

Etape 2: Met en place le gestionnaire de signaux sur le signal avec la fonction sigaction

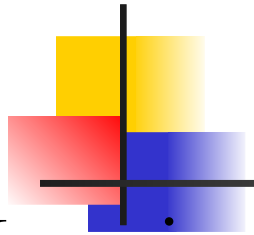
Etape 3: Un signal est émis ; le programme du processus est dérouté vers la fonction. A la fin du traitement, le processus reprend son programme.

Concurrence/synchronisation/ signaux



Les signaux norme POSIX Exemple

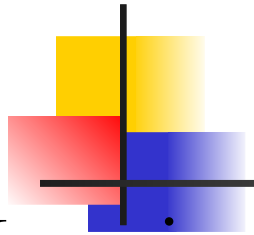
```
void     handler (int sig)
  printf("SIGNAL %d  recu\n",sig);
}
int main (){
  struct sigaction          actions;
  int rc;
  sigemptyset(&actions.sa_mask);
  actions.sa_flags = 0;
  actions.sa_handler = handler;
  // arme le signal
  rc = sigaction(SIGINT,&actions,NULL); //ctrl C
  getchar();
}
```



Concurrence/synchronisation/ signaux

Les signaux norme POSIX Exemple

```
 > a.out
Ctrl C
SIGNAL 2 recu
Ctrl C
SIGNAL 2 recu
Ctrl Z
 > bg
501 a.out
 > kill -INT 501    // ou kill -2 501
SIGNAL 2 recu
 > kill -KILL 501
501 killed a.out
```



Les signaux norme POSIX Exemple

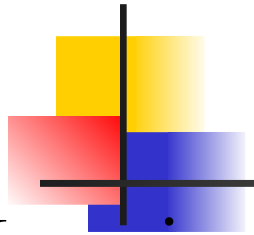
Le gestionnaire peut être utilisé pour plusieurs signaux.

```
..
rc = sigaction(SIGINT,&actions,NULL);
rc = sigaction(SIGSEGV,&actions,NULL);
rc = sigaction(SIGUSR1,&actions,NULL);



void     handler (int sig) {
}
sig indique quel signal a été  recu
```

Concurrence/synchronisation/ signaux

Les signaux norme POSIX Exemple USR1

```
void     fonction (int sig){
 printf("SIGNAL %d processus %d\n",sig,getpid());
 exit(1);   // SORTIE DU FILS
}
int main() {
  pid_t pid
  if   ((pid=fork()))==0) { //fils
    struct sigaction        actions;
    sigemptyset(&actions.sa_mask);
    actions.sa_flags = 0;
    actions.sa_handler = fonction;
    // arme le signal
    rc = sigaction(SIGUSR1,&actions,NULL);
    // tache
```

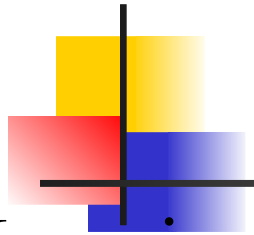


Les signaux norme POSIX Exemple USR1

```
    } else { //père
      // tache

      …
      kill (pid,SIGUSR1);
   }
}
```

Attention, Ici il faut que le fils est eu le temps de mettre en place son gestionnaire de signaux

Les fils lors d'un fork hérite du gestionnaire de signaux du père.



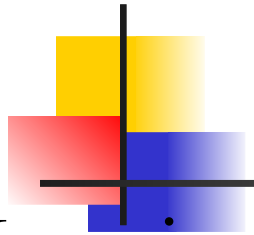
Les signaux norme POSIX

Comportement par défaut (Norme POSIX):

Le gestionnaire est automatiquement réarmé à la fin du traitement pour la réception d ’autres signaux du même type

Dans le gestionnaire de signaux, le processus peut recevoir d’autres signaux et donc empiler les gestionnaires de signaux.

On peut définir l’ensemble des signaux ne pouvant interrompre le traitement d’un signal et donc différer leurs traitements

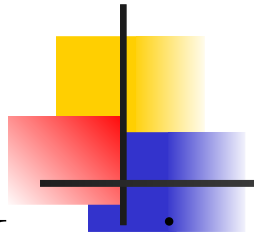


Les signaux norme POSIX

Action à effectuer à la réception d'un signal

La norme définit les ensembles de signaux

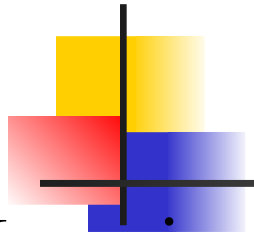
```
int sigemptyset(sigset_t *set);
          // vide set
int sigaddset(sigset_t *set,int sig);
          // add sig dans set
int sigdelset(sigset_t *set,int sig);
          // del sig de set
int sigfillset(sigset_t *set);
          // tous les signaux dans set
int sigismember(sigset_t *set,int sig);
          // 1 si sig est dans set
```



Les signaux norme POSIX

```
struct sigaction {
    void      (* sa_handler)   (int);
    void      (* sa_sigaction)
                  (int, siginfo_t *, void *);
    sigset_t     sa_mask;
    int          sa_flags;
    void      (* sa_restorer)  (void);//obsolet
}
```

sa_handler ou sa_sigaction définit un pointeur sur la fonction qui sera appelée à la réception d'un signal



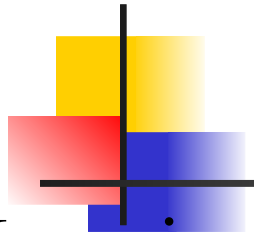
Concurrence/synchronisation/ signaux

Les signaux norme POSIX

```
struct sigaction {
    void      (* sa_handler)    (int);
    sigset_t     sa_mask;
    int          sa_flags;
}
```

version simple : sa_handler définit un pointeur sur la fonction que sera appelée à la réception d'un signal (gestionnaire de signaux)

sa_mask définit l'ensemble des signaux bloqués lors de l'appel de la fonction qui traite déjà un signal reçu.



Les signaux norme POSIX

sa_flags spécifie un ensemble d'attributs qui modifient le comportement du gestionnaire de signaux. Il est formé par un OU binaire (|) entre les options suivantes :

0

SA_ONESHOT (armé une seule fois le gestionnaire)

SA_RESTART Fournit un comportement compatible avec la sémantique BSD en redémarrant automatiquement les appels systèmes lents interrompus par l'arrivée du signal

SA_NOMASK ou SA_NODEFER Ne pas empêcher un signal d'être reçu depuis l'intérieur de son propre gestionnaire.

Les signaux norme POSIX

```
void     fonction (int sig){
 printf("SIGNAL %d \n",sig);
 sleep(60); // attente (mal choisi)
 printf("fin traitement du signal %d\n",sig);
}
int main() {
  struct sigaction        actions;
  int rc;
  sigemptyset(&actions.sa_mask);
  actions.sa_flags = 0;
  actions.sa_handler = handler;
  rc = sigaction(SIGUSR1,&actions,NULL);
 sigaddset( &actions.sa_mask,SIGUSR1);
  // diffère les usr1 lors du traitement des usr2
  rc = sigaction(SIGUSR2,&actions,NULL);
  getchar();
}
```

Les signaux norme POSIX

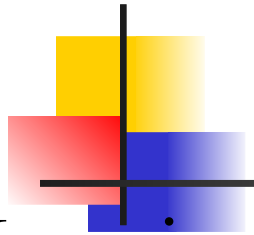
```
> a.out &
501 a.out
> kill -USR1 501
> kill -USR2 501
SIGNAL 30
SIGNAL 31
fin du traitement du signal 31
fin du traitement du signal 30
> a.out &
502 a.out
> kill -USR2 502
> kill -USR1 502
SIGNAL 31
fin du traitement du signal 31
SIGNAL 30
fin du traitement du signal 30
```

Les signaux norme POSIX

sigprocmask
définit la liste des signaux autorisés (ou mis en attente)
Paramètres :

SIG_SETMASK (position)SIG_UNBLOCK(retire) SIG_BLOCK(ajout)

```
..
sigset_t set;
sigemptyset (&set);
sigaddset (&set, SIGUSR1);
sigprocmask (SIG_SETMASK, &set, NULL);
//les signaux SIGUSR1 sont mis en attente
…
sigprocmask (SIG_UNBLOCK, &set, NULL);
//les signaux SIGUSR1 sont lus
```



Concurrence/synchronisation/signaux

Les signaux norme POSIX

sigprocmask
définit la liste des signaux autorisés (ou mis en attente)

```
..
sigset_t set;
sigset_t old_set;
sigemptyset (&set);
sigaddset (&set, SIGUSR1);
sigprocmask (SIG_SETMASK, &set, &old_set);
// les signaux SIGUSR1 sont mis en attente
// l'ancien mask est stocké dans old_set
```

Concurrence/synchronisation/signaux

Les signaux norme POSIX

Exemple : ignorer un signal

```
int main (){
   struct sigaction          actions;
   sigemptyset(&actions.sa_mask);
   actions.sa_flags = 0;
   actions.sa_handler = SIG_IGN;//Pas de Fonction
   sigaction(SIGINT,&actions,NULL);
   { /* tache courante */
   int c;
   while(1)
      read(0,&c,1);
   }
```

// les CTRL C sont ignorés

Les signaux norme POSIX

La primitive alarm(n) demande au noyau d ’envoyé un signal SIGALRM dans secondes. alarm(0) annule la demande

Exemple mis en place d ’un timeout

Les signaux norme POSIX

```
static void     fonction (int sig){
   fdatasync(0); //arret du getchar();
   return;
}
int main (){
  struct sigaction        actions;
  sigemptyset(&actions.sa_mask);
  actions.sa_flags = 0;
  actions.sa_handler = fonction;
  rc = sigaction(SIGALRM,&actions,NULL);
  alarm(5); // SIGALARM dans 5 secondes
  getchar();
  alarm(0); //annule l 'alarme
}
```

Concurrence/synchronisation/ signaux

Les signaux norme POSIX

settitimer(int which,struct itimerval*value, struct itimerval*oldvalue)

demande au noyau d'envoyer un signal à l 'expiration d 'un délai, Ce délai s'exprime en secondes et micro secondes.

Il y a trois temporisations par processus (which)

ITIMER_REAL (temps réel SIGALRM)

ITIMER_VIRTUAL (temps cpu du processus SIGVTALRM)

ITIMER_PROF (VIRTUAL + appel système SIGPROF)

Les timers peuvent être annulés avec une valeur 0;

La fonction getitimer (int which,struct itimerval*value) récupère la valeur actuelle des délais RESTANTS

Concurrence/synchronisation/signaux

Les signaux norme POSIX

La primitive sigsuspend(sigset_t set) suspend le processus courant sur l 'arrivée d'un signal autre que set

(modifie préalablement l'ensemble des signaux bloqués)

la primitive pause() suspend le processus sur n 'importe quel signal (non conforme POSIX)

Exemple : la fonction sleep de la libC :

```
static void     fonction (int sig){
 return;  //RAS}
int sleep (unisgned nb){
  struct sigaction          actions;
  sigset_t set,oset; // nouveau et ancien mask
  sigemptyset (&set);
  sigaddset (&set, SIGALRM); //uniquement SIGALRM
```

Concurrence/synchronisation/ signaux

Les signaux norme POSIX

```
// sauvegarde le mask courant dans oset
// on place le nouveau mask SIGALRM bloqué
sigprocmask (SIG_BLOCK, &set, &oset);
actions.sa_mask = oset;  //diffère les anciens
actions.sa_flags = 0;
actions.sa_handler = fonction;
sigaction(SIGALRM,&actions,NULL);
// Demande au noyau un signal dans nb secondes
alarm(nb);
// pause
sigsuspend(&oset); // suspend jusqu'au SIGALRM
        // tout sauf les anciens signaux
// remet l 'ancien mask
sigprocmask (SIG BLOCK, &oset, NULL);
```

Les signaux norme POSIX

```
struct sigaction {
    void     (* sa_handler)   (int);
    void      (* sa_sigaction)
                (int, siginfo_t *, void *);
    sigset_t    sa_mask;
    int         sa_flags; // SA_SIGINFO
    void     (* sa_restorer)  (void);//obsolet
}
```

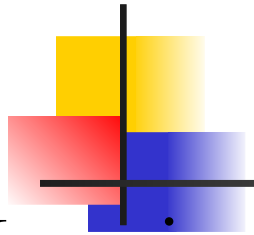
sa_sigaction définit un pointeur sur la fonction que sera appelée à la réception d'un signal

Concurrence/synchronisation/ signaux

Les signaux norme POSIX

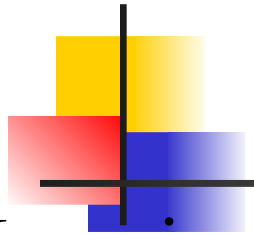
```
siginfo_t {
  int     si_signo;    // Numéro de signal
  int     si_errno;    // Numéro d'erreur
  int     si_code;     // Code du signal
  pid_t   si_pid;      // PID de l'émetteur
  uid_t   si_uid;      // UID réel de l'émetteur
  int     si_status;  // Valeur de sortie
  clock_t si_utime;   //Temps utilisateur écoulé
  clock_t si_stime;   // Temps système écoulé
  sigval_t si_value;  // Valeur de signal
  int     si_int;     // Signal Posix.1b
  void *  si_ptr;     // Signal Posix.1b
  void *  si_addr;    // Emplacement d'erreur
  int     si_band;    // Band event
  int     si_fd;      // Descripteur de fichier

              }
```

Concurrence/synchronisation/ signaux



Les signaux norme POSIX

```
static void     fonction (int sig,siginfo_t *
  info,void * arg){
  printf("SIGNAL %d \n",sig);
  if (info) {
    printf("info->si_signo %d\n",info->si_signo);
    printf("info->si_code %d\n",info->si_code);
    printf("info->si_pid %d\n",info->si_pid);
    printf("info->si_uid %d\n",info->si_uid);
    printf("info-si_status%d\n",info->si_status);
    printf("info->si_int %d\n",info->si_int);
    printf("info->si_band %d\n",info->si_band);
  }
```



Les signaux norme POSIX

```
int main (){
  struct sigaction        actions;
  int rc;
  sigemptyset(&actions.sa_mask);
  actions.sa_flags = SA_SIGINFO;
  actions.sa_sigaction = fonction;
  rc = sigaction(SIGINT,&actions,NULL);
  printf("Effectuer un Ctl C ou \
          kill -INT %d pid\n",getpid());
  getchar();
```