

# Noyau d'un système d'exploitation

## INF2610

### Séances d'exercices



Département de génie informatique et génie logiciel



Automne 2017

# Séance d'exercices

## Moniteurs

## Gestion de la mémoire

## Windows

## Interblocage

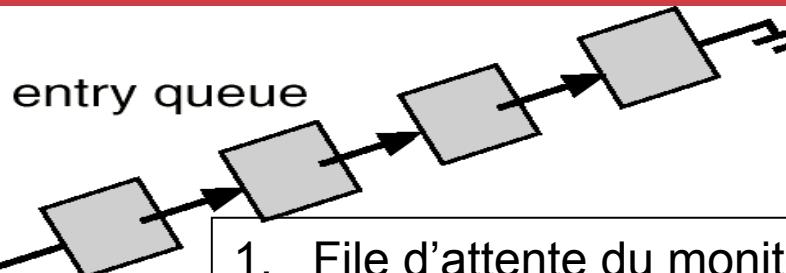
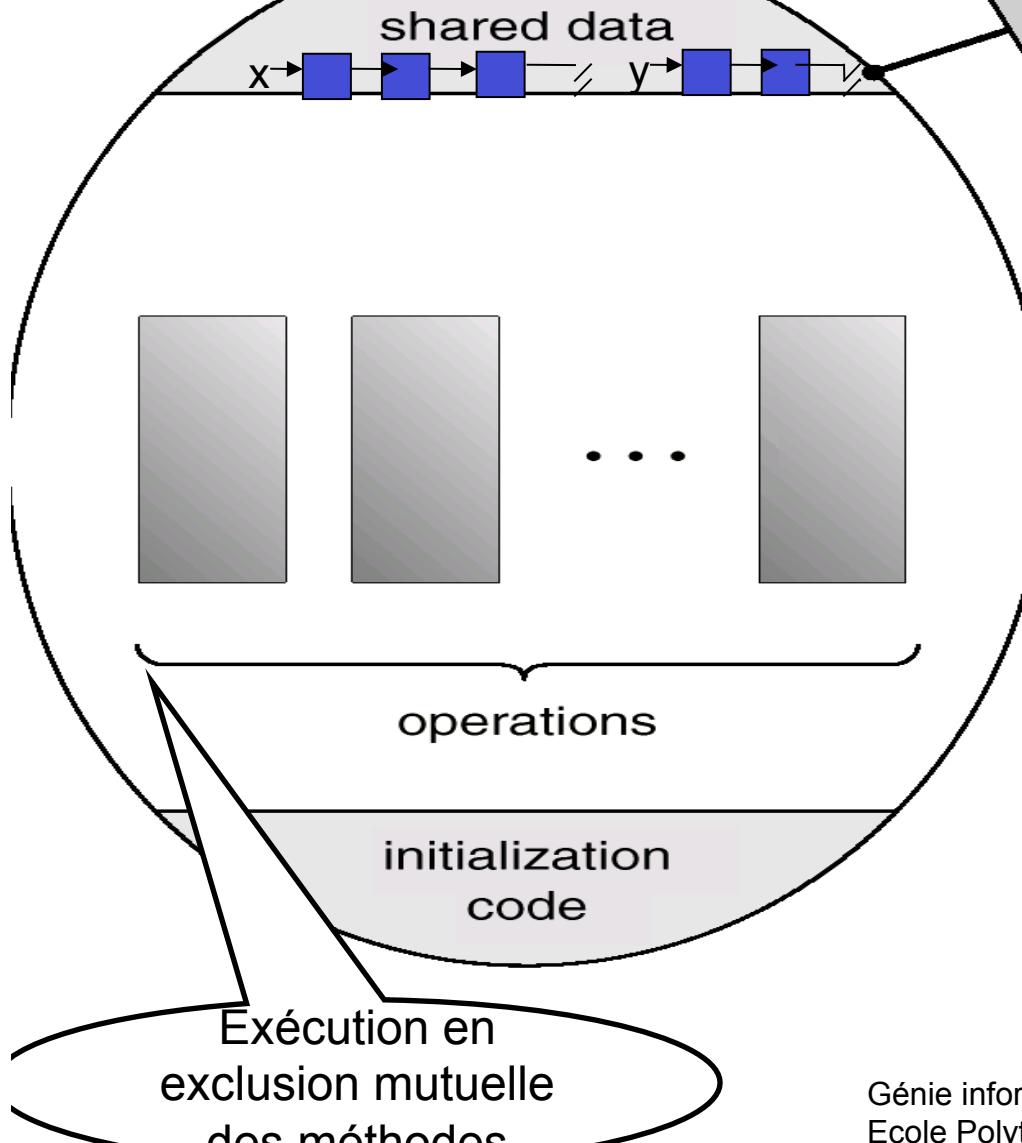
## Ordonnancement classique et ordonnancement temps réel



<http://www.cse.psu.edu/~gxt29/bug/softwarebug.html>  
<https://nvd.nist.gov/>

# Moniteurs et variables de condition (signal-and-continuer ou signal-and-wait)

queues  
associated with  
x,y conditions



1. File d'attente du moniteur.
2. Deux files d'attente pour x et y.
3. File d'attente des processus suspendus (cas de signal-and-wait).

Variables de condition x et y :  
wait(x) : se bloquer dans le moniteur (après avoir réveiller un processus en attente du moniteur). Le processus se retrouve dans la file d'attente de x.

signal(x) : débloque un processus de la file d'attente de x. Deux sémantiques :

- **Signal-and-continuer** : Le processus débloqué est inséré dans la file d'attente du moniteur.
- Signal-and-wait : Le processus appelant signal est inséré dans la file d'attente de processus suspendus dans le moniteur. Le processus débloqué devient actif dans le

# Moniteurs et variables de condition

Comment utiliser les moniteurs pour assurer l'exclusion mutuelle ?

Moniteur Compte

```
{ int solde = 0 ;
```

```
void Deposer (int montant) // section critique pour le dépôt
```

```
{   solde = solde + montant ;  
}
```

```
void retirer (int montant) //section critique pour le retrait
```

```
{   if (solde >= montant)  
       solde = solde - montant ;  
}  
}
```



# Moniteurs et variables de condition

## Problème Producteur/consommateur

Moniteur ProducteurConsommateur (int N)

```
{  boolc nplein, nvide ;  
    int tampon[N], compteur =0, ic=0, ip=0 ;  
    void mettre (int objet)  
    { if (compteur==N) wait(nplein) ;  
        tampon[ip] = objet ;  
        ip = (ip+1)%N ;  
        compteur++ ;  
        if (compteur==1) signal(nvide) ;  
    }  
    void retirer (int& objet)  
    { if (compteur ==0) wait(nvide) ;  
        objet = tampon[ic] ;  
        ic = (ic+1)%N ;  
        compteur -- ;  
        if (compteur==N-1) signal(nplein) ;  
    }  
}
```

Doit-on modifier le code pour le cas de plusieurs producteurs et plusieurs consommateurs ?



# Moniteurs et variables de condition

## Problème Producteur/consommateur

```
Moniteur ProducteurConsommateur (int N)
{
    boolc nplein, nvide ;
    int tampon[N], compteur =0, ic=0, ip=0 ;
    void mettre (int objet)
    { if (compteur==N) wait(nplein) ;
        tampon[ip] = objet ;
        ip = (ip+1)%N ;
        compteur++ ;
        if (compteur==1) signal(nvide) ;
    }
    void retirer (int& objet)
    { if (compteur ==0) wait(nvide) ;
        objet = tampon[ic] ;
        ic = (ic+1)%N ;
        compteur -- ;
        if (compteur==N-1) signal(nplein) ;
    }
}
```

- Cas de 2 producteurs (P1, P2), un consommateur C et N=1:
  - P1: mettre(o1): tampon[0]=o1; ip=0; compteur=1, signal(nvide); P1 quitte le moniteur.
  - P2: mettre(o2): rentre dans le moniteur.
  - Pendant P2 est dans le moniteur C: retirer(o); puis P1.mettre(o1); → C et P1 sont mis en attente du moniteur.
  - P2: mettre(o2): wait(nplein) → P2 est mis en attente de nplein et C rentre dans le moniteur.
  - C : retirer(o) : o=tampon[0]; ic=0;compteur=0; signal(nplein) => P2 est mis en attente du moniteur. C quitte le moniteur et P1 rentre dans le moniteur.
  - P1 : mettre(o1) : tampon[0]=o1; ip=0; compteur=1, signal(nvide); P1 quitte le moniteur.
  - **P2: tampon[1]=o1; ip=0; compteur=2,**

Doit-on modifier le code pour le cas de plusieurs producteurs et plusieurs consommateurs ?



# Moniteurs et variables de condition

## Problème Producteur/consommateur

```
Moniteur ProducteurConsommateur (int N)
{
    boolc nplein, nvide ;
    int tampon[N], compteur =0, ic=0, ip=0 ;
    void mettre (int objet)
    { while (compteur==N) wait(nplein) ;
        tampon[ip] = objet ;
        ip = (ip+1)%N ;
        compteur++ ;
        if (compteur==1) signal(nvide) ;
    }
    void retirer (int& objet)
    { while (compteur ==0) wait(nvide) ;
        objet = tampon[ic] ;
        ic = (ic+1)%N ;
        compteur -- ;
        if (compteur==N-1) signal(nplein) ;
    }
}
```

- Cas de 2 producteurs (P1, P2), un consommateur C et N=1.
- P1: mettre(o1): tampon[0]=o1; ip=0; compteur=1, signal(nvide); P1 quitte le moniteur.
- P2: mettre(o2): rentre dans le moniteur.
- Pendant P2 est dans le moniteur C: retirer(o); puis P1.mettre(o1); **➔** C et P1 sont mis en attente du moniteur.
- P2: mettre(o2): wait(nplein) **➔** P2 est mis en attente de nplein et C rentre dans le moniteur.
- C : retirer(o) : o=tampon[0]; ic=0;compteur=0; signal(nplein) => P2 est mis en attente du moniteur. C quitte le moniteur et P1 rentre dans le moniteur.
- P1 : mettre(o1) : tampon[0]=o1; ip=0; compteur=1, signal(nvide); P1 quitte le moniteur.
- **P2: wait(nplein) ➔ P2 est mis en attente.**

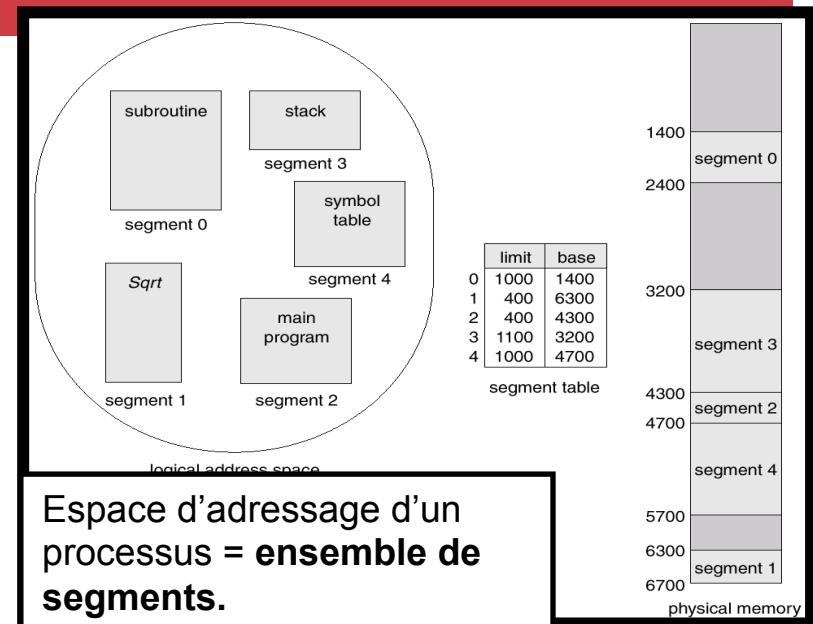
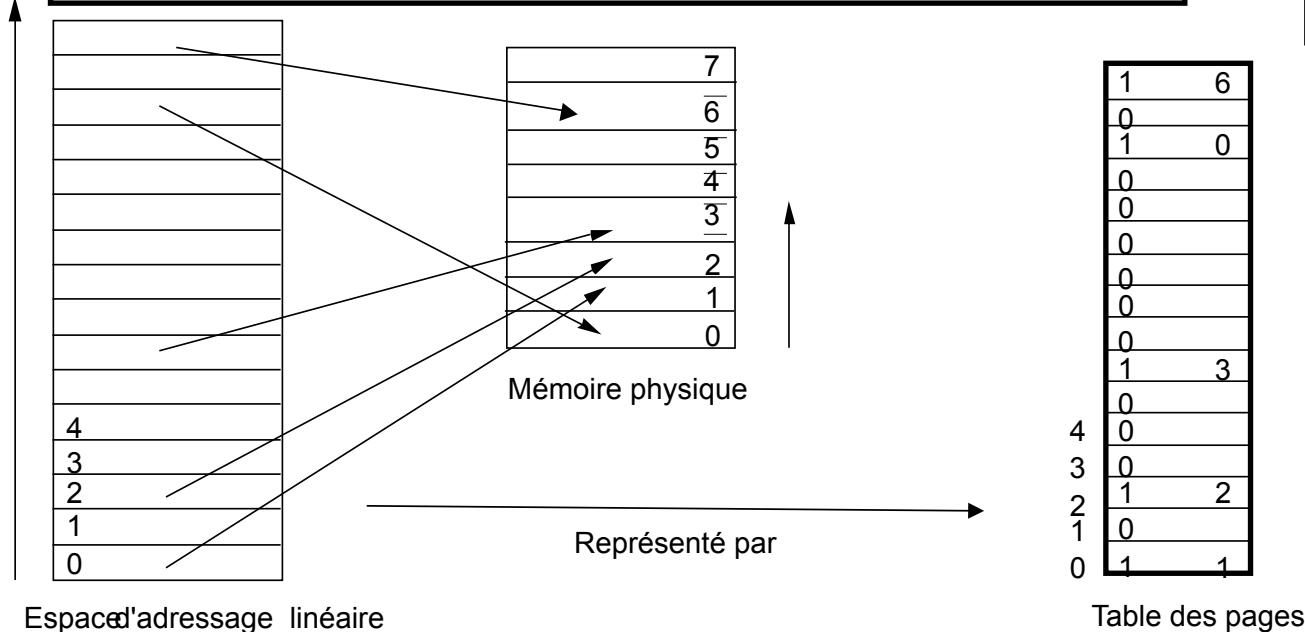


Doit-on modifier le code pour le cas de plusieurs producteurs et plusieurs consommateurs ?

# Gestion de la mémoire

- Chaque processus a son propre espace d'adressage et accède à la mémoire physique via cet espace.

Espace d'adressage d'un processus = **ensemble de pages**. Taille d'une page = taille d'un cadre.

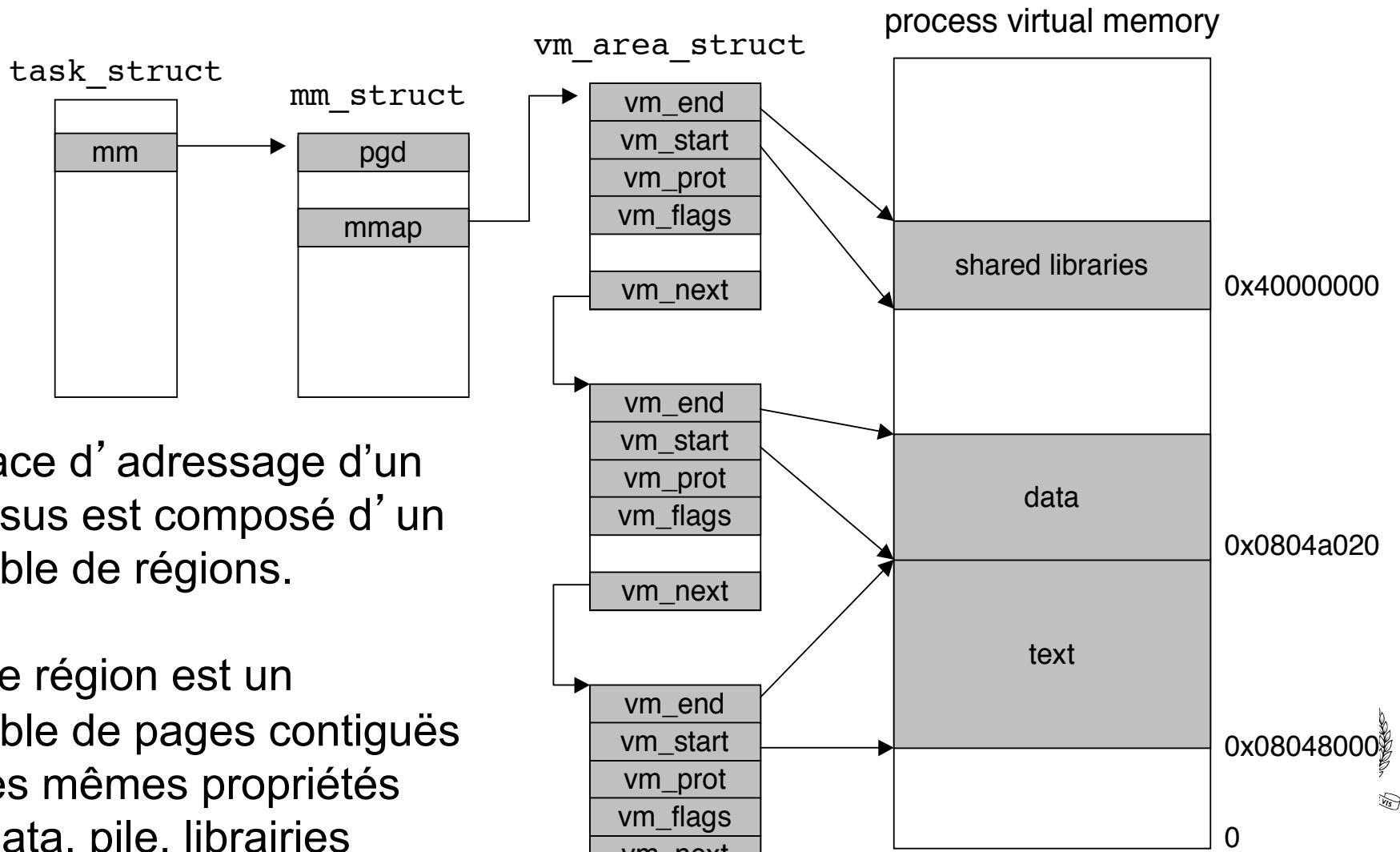


Espace d'adressage d'un processus = **ensemble de segments**.

Espace d'adressage d'un processus = **ensemble de segments**.  
Segment = **ensemble de pages de même taille**.  
=> Un adressage linéaire

# Gestion de la mémoire

## Espace d'adressage d'un processus (cas de Linux)

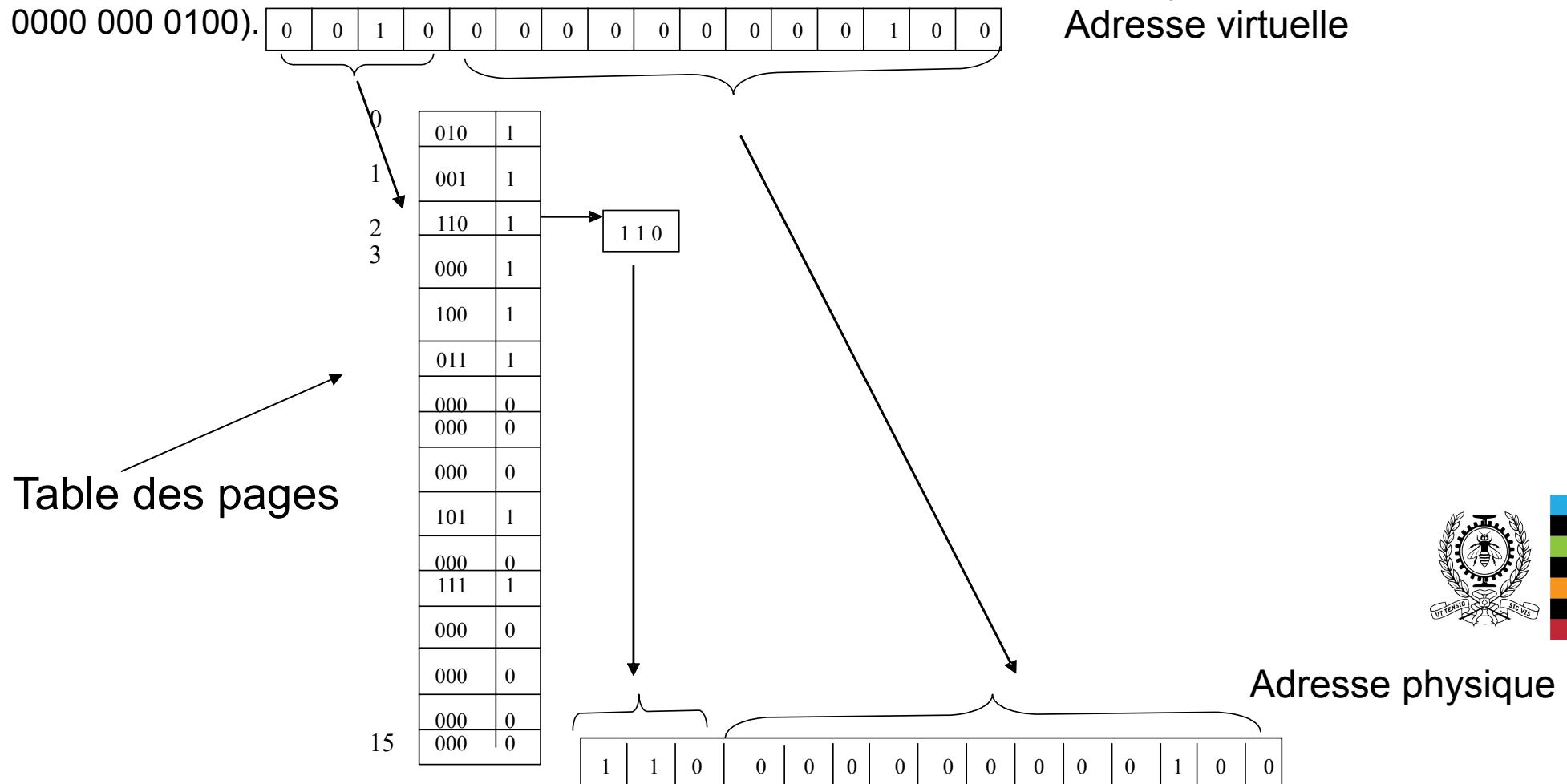


- L'espace d'adressage d'un processus est composé d'un ensemble de régions.
- Chaque région est un ensemble de pages contiguës avec les mêmes propriétés (text, data, pile, librairies partagées, fichiers mappés, etc.).

# Gestion de la mémoire

## Conversion d'adresses virtuelles (pagination pure)

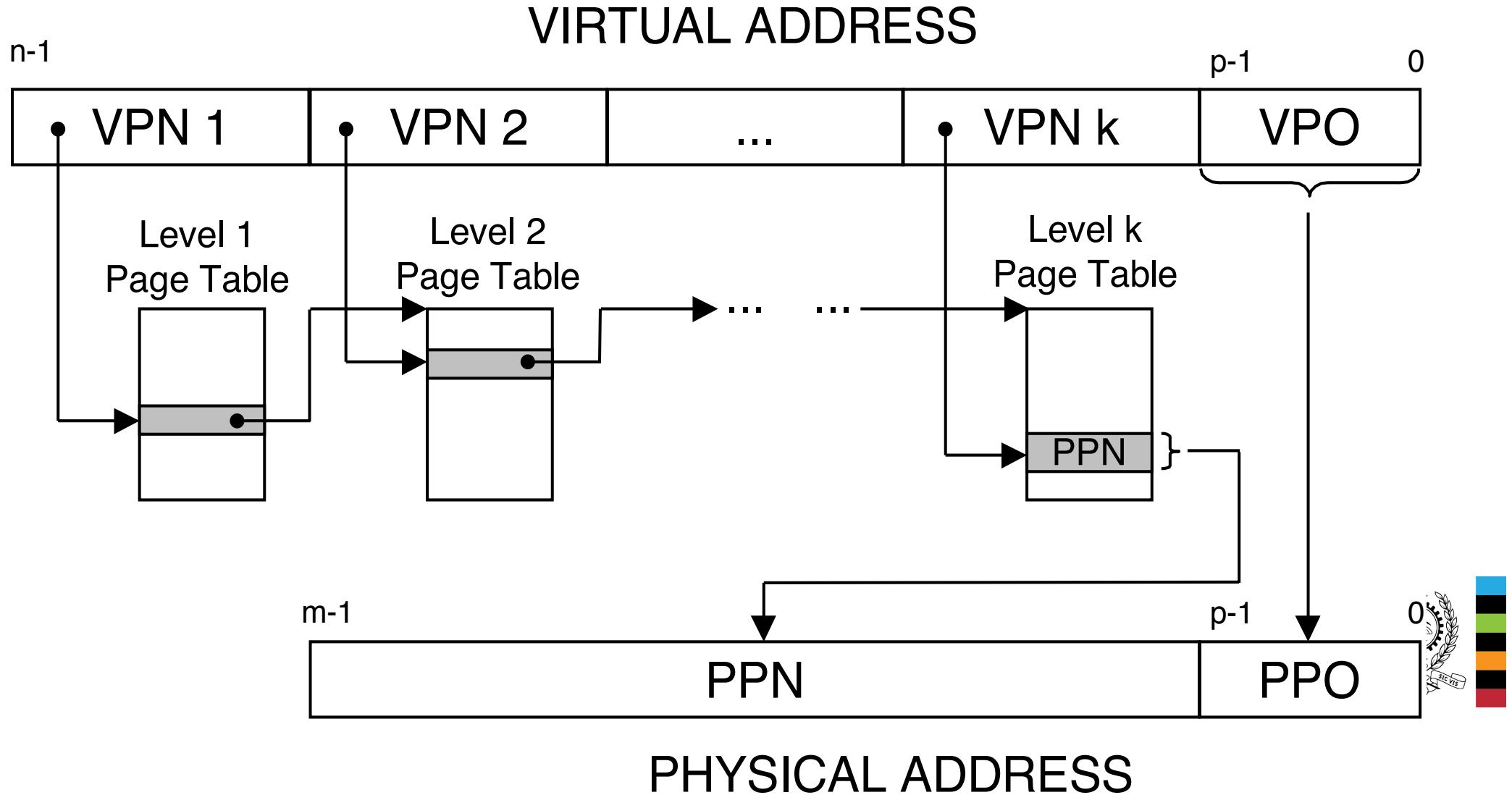
- Adresse virtuelle = (numéro de page, déplacement dans la page).
- Chaque entrée de la table des pages est composée de plusieurs champs, notamment : bit de présence (P), bit de référence (R), bits de protection (un, deux ou trois bits), bit de modification (M appelé bit dirty/clean), numéro de case correspondant à la page et son emplacement sur disque
- L'adresse virtuelle 8196 (0010 0000 0000 0100) est convertie en adresse physique 24580 (110 0000 000 0100).



# Gestion de la mémoire

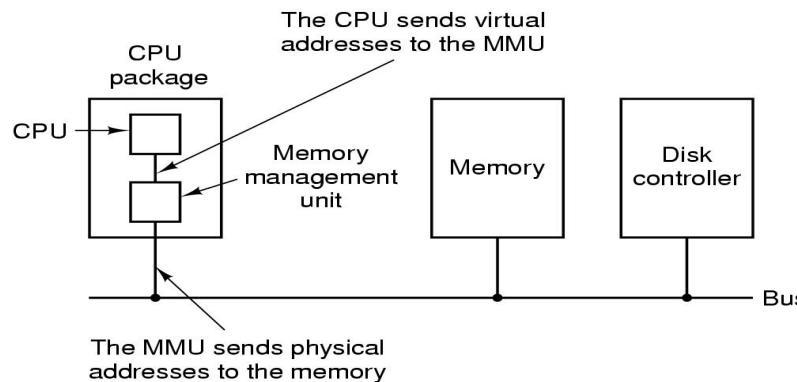
## Conversion d'adresses virtuelles (pagination pure)

### Table de pages à k niveaux



# Gestion de la mémoire

## Conversion d'adresses virtuelles (pagination pure)

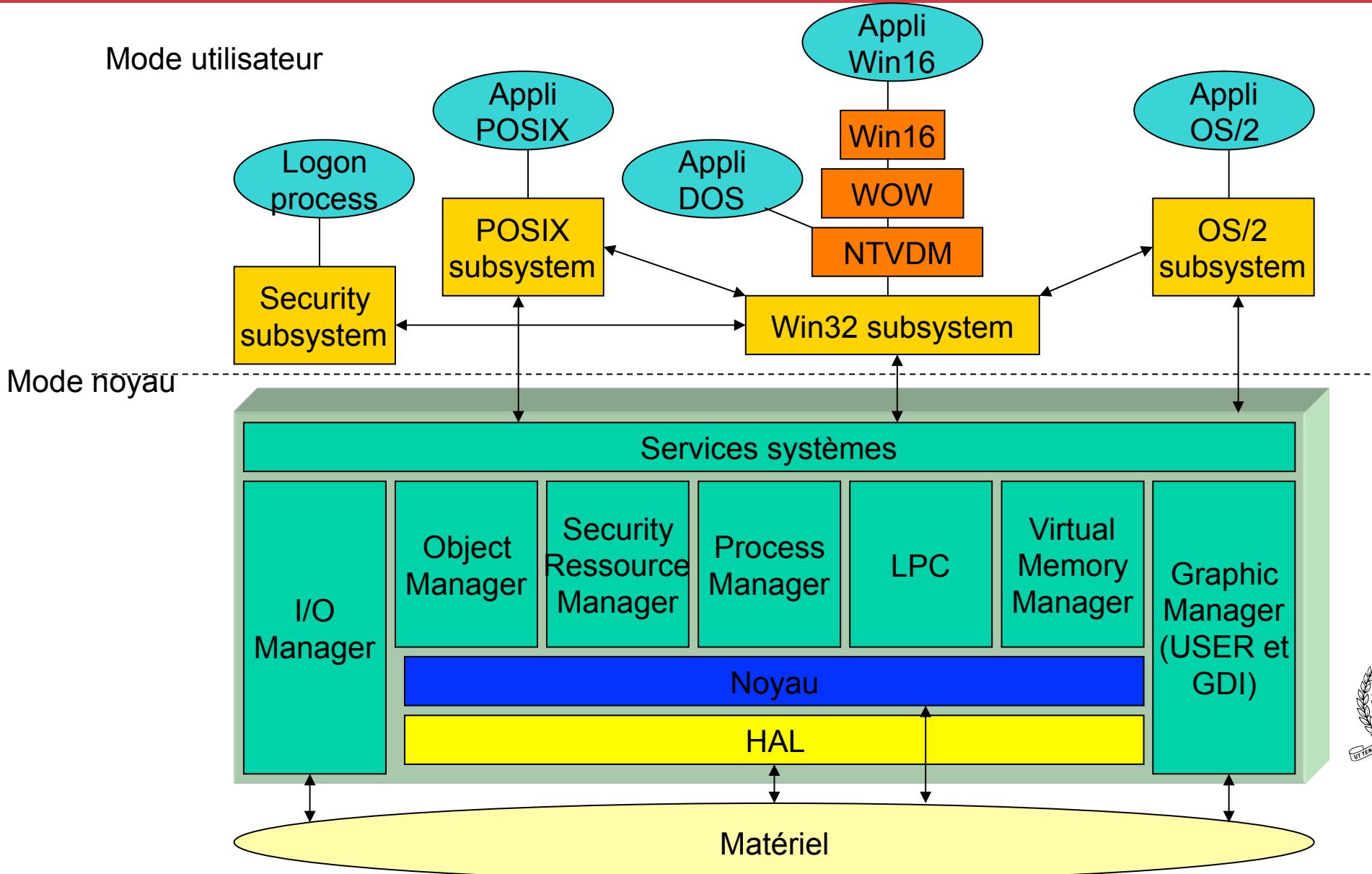


1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

- Algorithmes de remplacement de page (manque d'espace mémoire)
- Le choix de la page à retirer dépend des références passées : LRU, Horloge, FIFO, Aléatoire (et Optimal).
- En cas de défaut de page, chargement de plusieurs pages voisines en mémoire.
- Allocation effective peut être retardée jusqu'à l'accès en écriture.



# Windows



<http://www-igm.univ-mlv.fr/~dr/XPOSE/archiNT/winnt4.html#NativeAPI>

# Windows

## Objets du système

Type	Description
Process	User process
Thread	Thread within a process
Semaphore	Counting semaphore used for interprocess synchronization
Mutex	Binary semaphore used to enter a critical region
Event	Synchronization object with persistent state (signaled/not)
Port	Mechanism for interprocess message passing
Timer	Object allowing a thread to sleep for a fixed time interval
Queue	Object used for completion notification on asynchronous I/O
Open file	Object associated with an open file
Access token	Security descriptor for some object
Profile	Data structure used for profiling CPU usage
Section	Structure used for mapping files onto virtual address space
Key	Registry key
Object directory	Directory for grouping objects within the object manager
Symbolic link	Pointer to another object by name
Device	I/O device object
Device driver	Each loaded device driver has its own object



# Windows

## Processus et threads

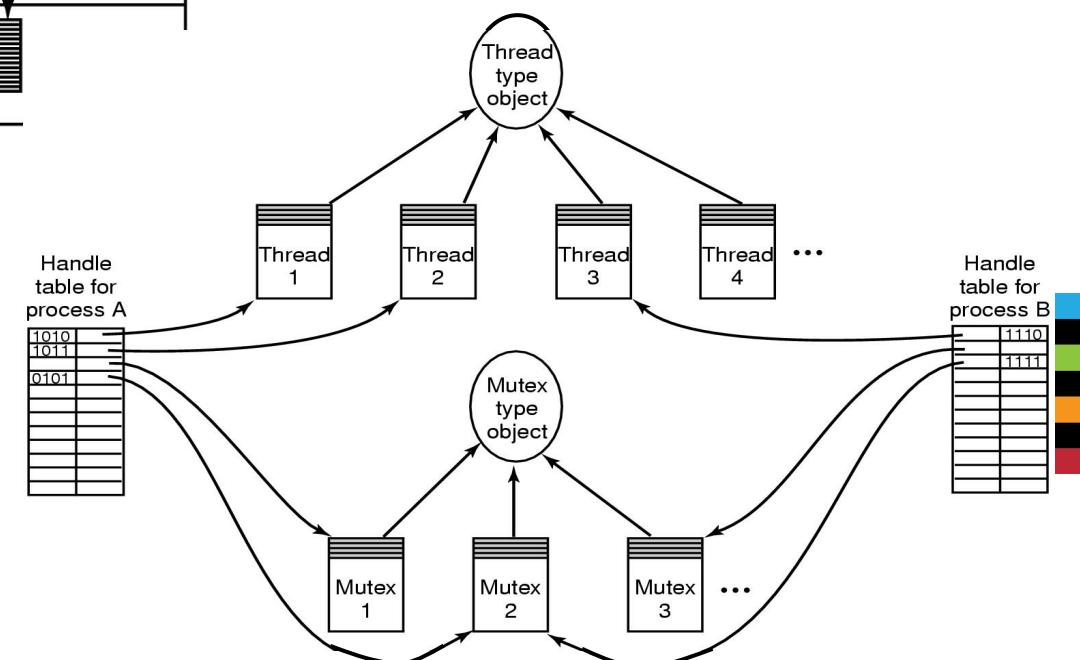
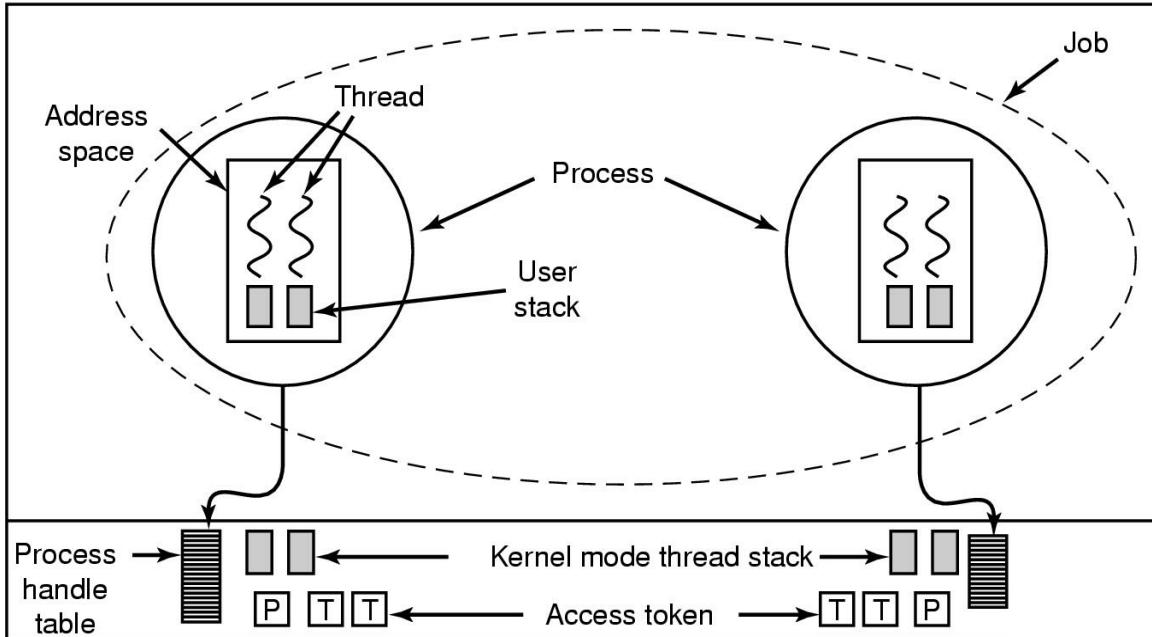
Win32 API Function	Description
CreateProcess	Create a new process
CreateThread	Create a new thread in an existing process
CreateFiber	Create a new fiber
ExitProcess	Terminate current process and all its threads
ExitThread	Terminate this thread
ExitFiber	Terminate this fiber
SetPriorityClass	Set the priority class for a process
SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semaphore
CreateMutex	Create a new mutex
OpenSemaphore	Open an existing semaphore
OpenMutex	Open an existing mutex
WaitForSingleObject	Block on a single semaphore, mutex, etc.
WaitForMultipleObjects	Block on a set of objects whose handles are given
PulseEvent	Set an event to signaled then to nonsignaled
ReleaseMutex	Release a mutex to allow another thread to acquire it
ReleaseSemaphore	Increase the semaphore count by 1
EnterCriticalSection	Acquire the lock on a critical section
LeaveCriticalSection	Release the lock on a critical section



CreatePipe, CreateNamedPipe, CreateFile, ReadFile, WriteFile, CloseHandle

# Windows Processus et threads

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms684161\(v=vs.85\).aspx#creating\\_jobs](https://msdn.microsoft.com/en-us/library/windows/desktop/ms684161(v=vs.85).aspx#creating_jobs)



# Windows

## Processus et threads

```
CreateProcess("fils.exe", //name or path of executable  
    NULL, // no command line.  
    NULL, // Process handle not inheritable.  
    NULL, // Thread handle not inheritable.  
    FALSE, // Set handle inheritance to FALSE.  
    0, // No creation flags.  
    NULL, // Use parent's environment block.  
    NULL, // Use parent's starting directory.  
    &si, // Pointer to STARTUPINFO structure.  
    &pi ) // Pointer to PROCESS_INFORMATION structure
```

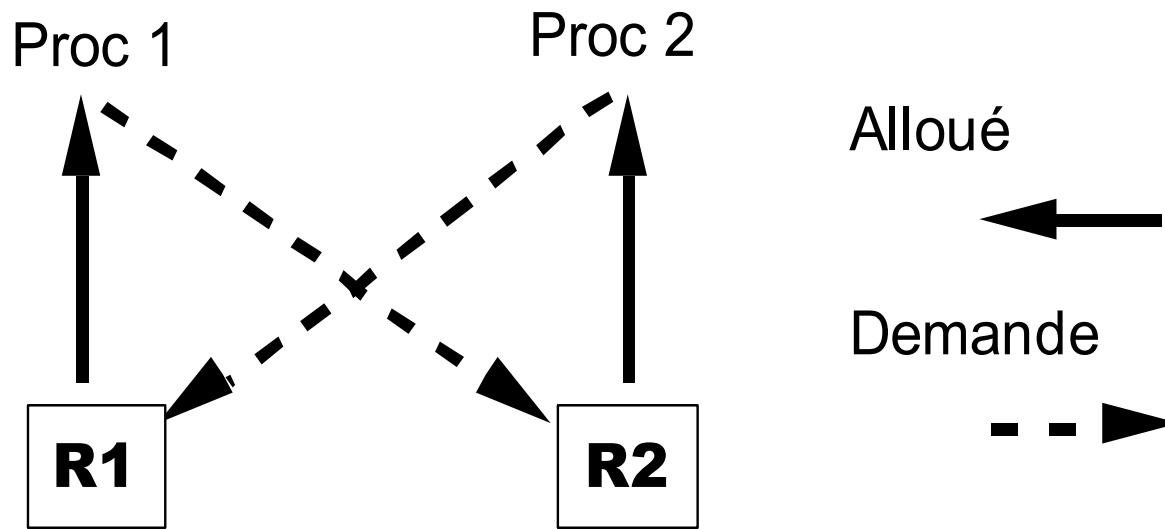
```
typedef struct _PROCESS_INFORMATION {  
    HANDLE hProcess;  
    HANDLE hThread;  
    DWORD dwProcessId;  
    DWORD dwThreadId;  
} PROCESS_INFORMATION,  
*LPPROCESS_INFORMATION;
```

```
typedef struct _STARTUPINFO {  
    DWORD cb;  
    LPTSTR lpReserved;  
    LPTSTR lpDesktop;  
    LPTSTR lpTitle;  
    DWORD dwX;  
    DWORD dwY;  
    DWORD dwXSize;  
    DWORD dwYSize;  
    DWORD dwXCountChars;  
    DWORD dwYCountChars;  
    DWORD dwFillAttribute;  
    DWORD dwFlags;  
    WORD wShowWindow;  
    WORD cbReserved2;  
    LPBYTE lpReserved2;  
    HANDLE hStdInput;  
    HANDLE hStdOutput;  
    HANDLE hStdError;  
} STARTUPINFO, *LPSTARTUPINFO;
```



# Interblocage

Processus qui partagent des ressources et s'exécutent en concurrence



1. Exclusion mutuelle : une ressource est soit allouée à un seul processus, soit disponible.
2. Détection et attente : les processus qui détiennent des ressources peuvent en demander d'autres.
3. pas de réquisition : les ressources allouées à un processus sont libérées uniquement par le processus (ressources non préemptives).
4. Attente circulaire: un ensemble de processus attendant chacun une ressource allouée à un autre.



# Solutions au problème d'interblocage

## La détection et la reprise

- Construire dynamiquement le graphe d'allocation des ressources.
- Ce graphe indique pour chaque processus, les ressources qu'il détient ainsi que celles qu'il demande.
- La détection est réalisée en réduisant le graphe (existence d'un ordonnancement qui permet à tous les processus de se terminer).

## L'évitemen~~t~~

- Dans ce cas, lorsqu'un processus demande une ressource, le système doit déterminer si l'attribution de la ressource est sûre (mènent vers un état sûr).
- Si c'est le cas, il lui attribue la ressource. Sinon, la ressource n'est pas accordée.
- Un état est sûr si tous les processus peuvent terminer leur exécution dans le pire cas (il existe un ordonnancement qui permet à tous les processus de se terminer) → **Algorithme du banquier**.
- Il faut connaître à l'avance les besoins en ressources de chaque processus.



# Solutions au problème d'interblocage

## La prévention des interblocages

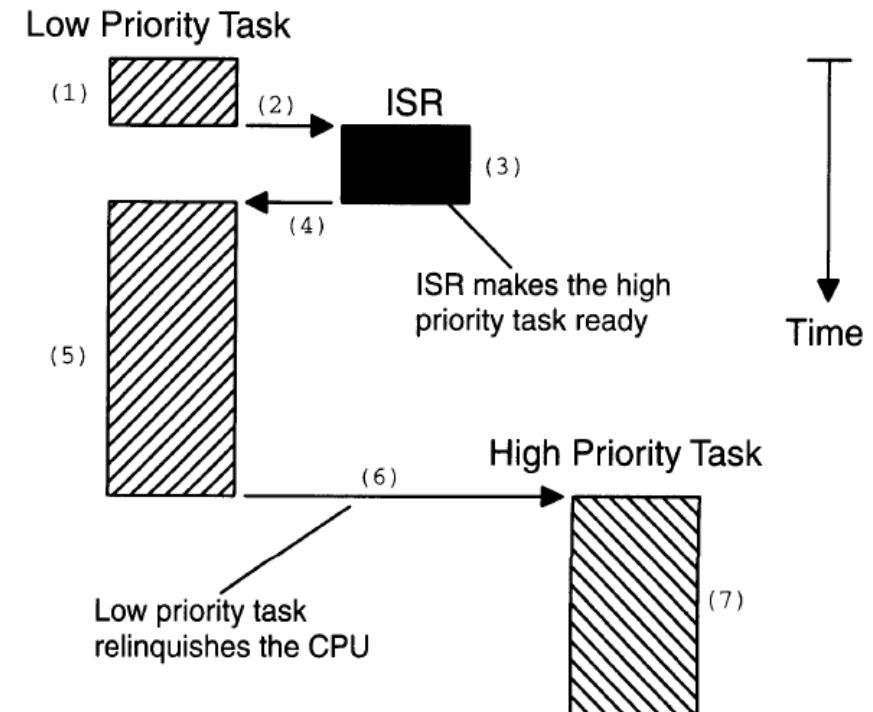
Pour prévenir les interblocages, il suffit de garantir que l'une des quatre conditions nécessaires à leur existence n'est jamais satisfaite.

- Pas d'exclusion mutuelle : impossible car certaines ressources sont à usage exclusif.
- Pas de « détention et attente » : Il faudrait que toutes les ressources nécessaires à un processus soient demandées et allouées à la fois. Le processus ne doit pas détenir des ressources et en demander d'autres.
  - Il est difficile de prévoir les besoins du processus
  - Problème de famine.
- Préemption : n'est pas raisonnablement traitable pour la plupart des ressources sans dégrader profondément le fonctionnement du système. On peut cependant l'envisager pour certaines ressources dont le contexte peut être sauvegardé et restauré.
- Pas d'attente circulaire : Enfin, on peut résoudre le problème de l'attente circulaire en numérotant les ressources et en n'autorisant leur demande, par un processus, que lorsqu'elles correspondent à des numéros croissants.



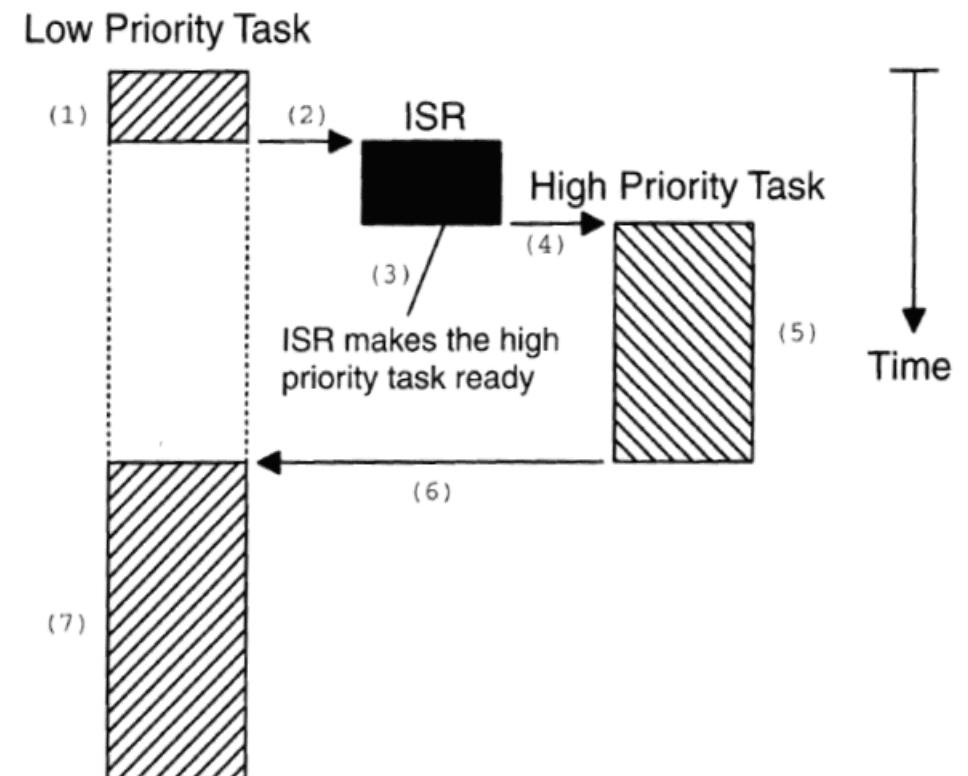
# Ordonnancement de processus non préemptif

- Le système d'exploitation choisit le prochain processus à exécuter :
  - Premier arrivé, Premier servi (FCFS, First-Come First-Served) ou
  - Plus court d'abord (SPF, Short Process First ou SJF Short Job First).
  - Plus prioritaire d'abord (priorité = (temps d'attente + temps d'exécution) / temps d'exécution).
- Il lui alloue le processeur jusqu'à ce qu'il se termine, se bloque (en attente d'un événement) ou cède le processeur. Il n'y a pas de réquisition.



# Ordonnancement de processus préemptif

- Suspendre le processus en cours -> fin d'un quantum, arrivée d'un processus plus prioritaire, etc. :
    - Plus court temps résiduel (Shortest Remaining Time)
    - Ordonnanceur circulaire (Round Robin)
    - Ordonnanceur à priorités et files multiples.
  - Ordonnancement à base de priorités → Problèmes :
    - Famine
    - Inversion des priorités
- => Priorités dynamiques
- => Quanta variables



# Ordonnancement temps réel (préemptif à priorités) de tâches périodiques indépendantes

- Un ensemble de n tâches T<sub>1</sub>,...,T<sub>n</sub> périodiques indépendantes (pas de sections critiques).

## Rate monotonic priority assignment (RMA)

- La priorité d'une tâche est inversement proportionnelle à sa période → Priorité statique.
- D<sub>i</sub> = P<sub>i</sub> pour i=1,n.
- Ordonnable si (condition suffisante de *Liu et Layland*) :

$$\sum_{i=1}^n \frac{C_i}{P_i} \leq n(2^{1/n} - 1)$$

n → ∞ => borne = 69.3 %

U<sub>i</sub> = C<sub>i</sub> / P<sub>i</sub> Taux d'occupation processeur de la tâche T<sub>i</sub>.

n	Utilization Bound
1	100.0%
2	82.8%
3	78.0%
4	75.7%
5	74.3%
10	71.8%

## Deadline monotonic priority assignment (DMA)

- La priorité d'une tâche est inversement proportionnelle à son deadline → Priorité statique
- Ordonnable si

$$\sum_{i=1}^n \frac{C_i}{D_i} \leq n(2^{\frac{1}{n}} - 1)$$



# Ordonnancement temps réel (préemptif à priorités) de tâches indépendantes

## Earliest Deadline First (EDF)

- La tâche la plus prioritaire (parmi les tâches prêtes) est celle dont l'échéance absolue est la plus proche → priorité dynamique.
- Il est applicable aussi bien pour des tâches périodiques qu'apériodiques.

• Ordonnancable si :

$$\sum_{i=1}^n \frac{C_i}{D_i} \leq 1$$

• Si  $P_i = D_i$  pour  $i=1,n$  → Ordonnancable ssi

$$\sum_{i=1}^n \frac{C_i}{D_i} \leq 1$$

## Tâches périodiques dépendantes → problème d'inversion de priorités

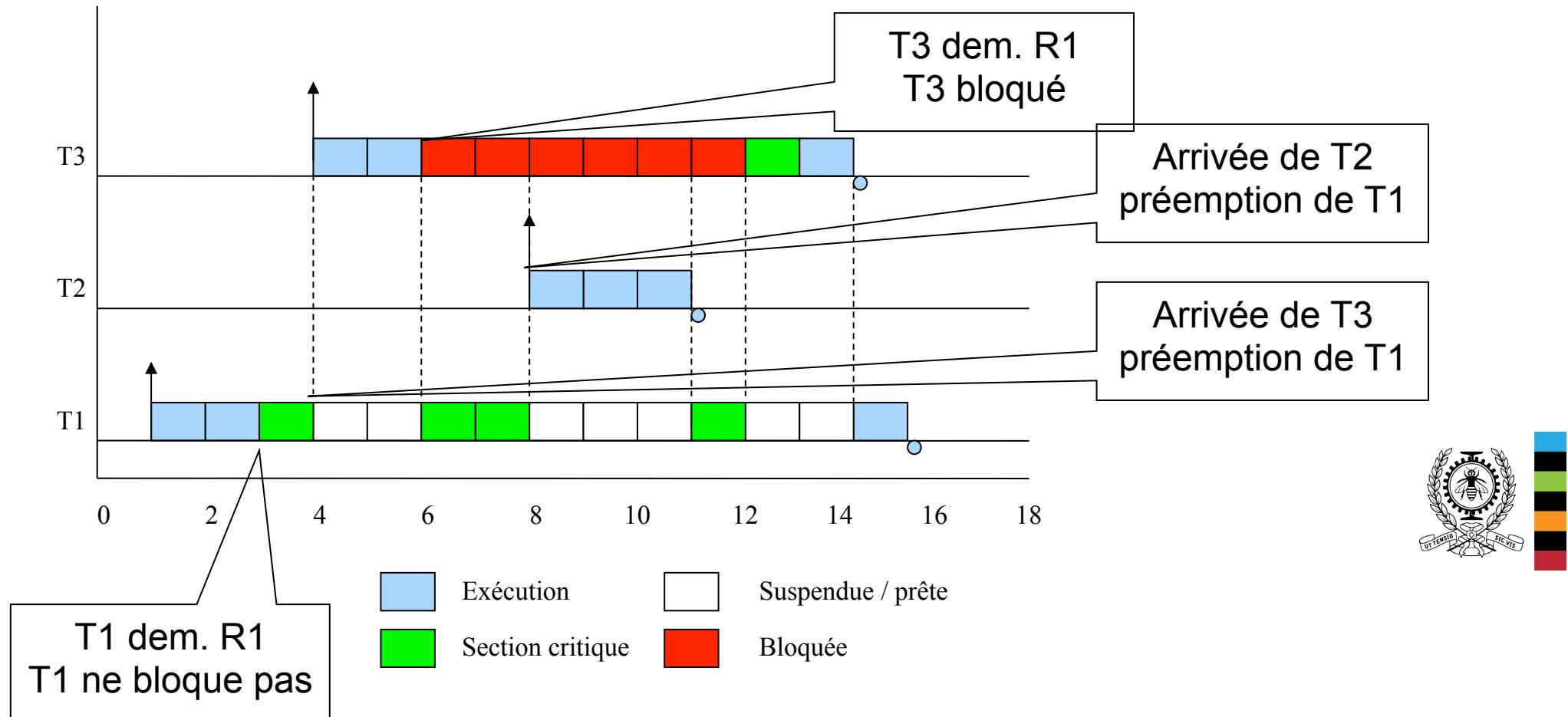
- Une tâche de basse priorité empêche une tâche plus prioritaire d'accéder à sa section critique ( bloque une tâche plus prioritaire).
- ⇒ Protocole PIP pour limiter la durée de l'inversion de priorités



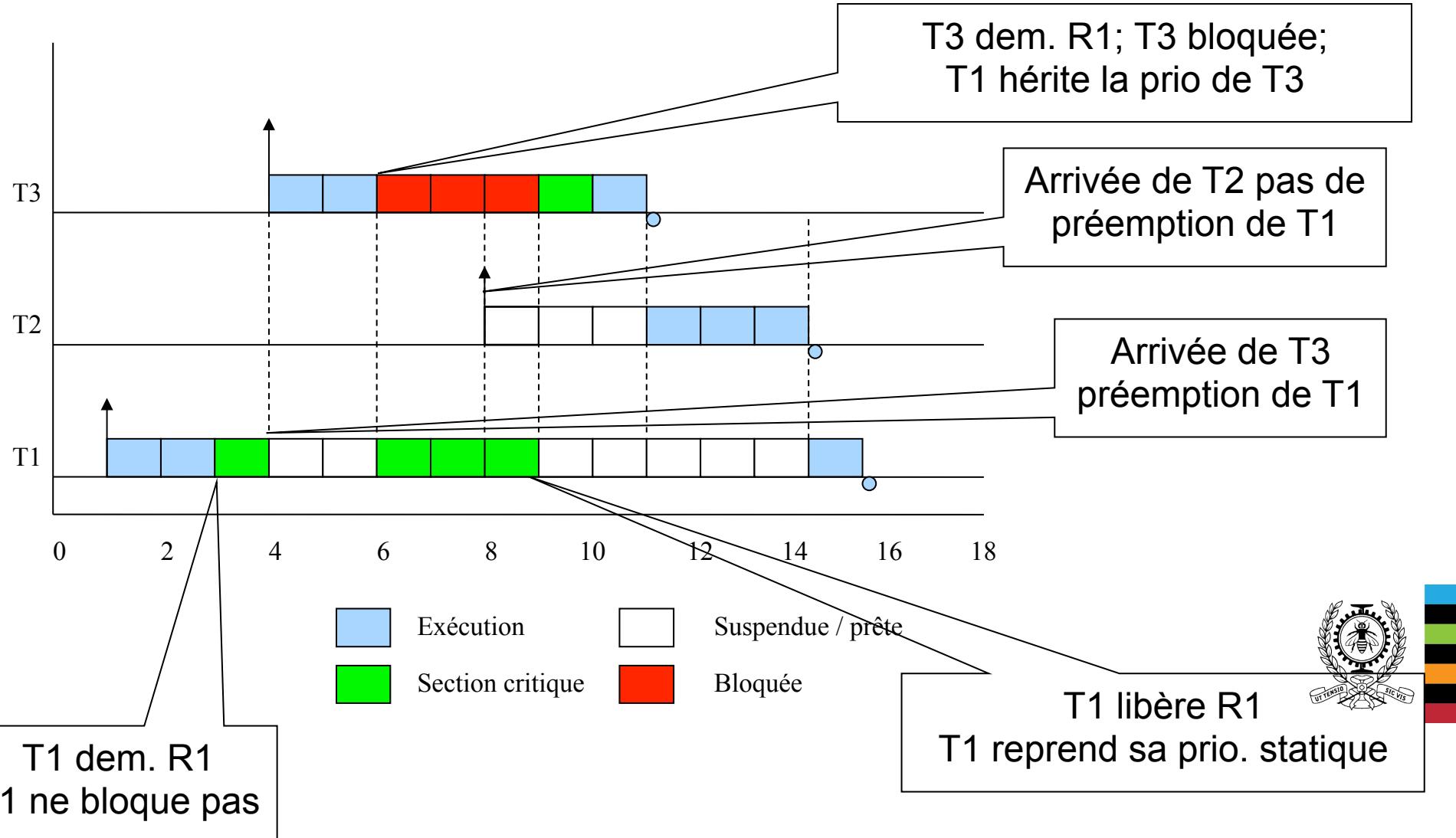
# Ordonnancement temps réel (préemptif à priorités) de tâches dépendantes → Inversion de priorités

Exemple :

Tâche	Date de départ	Priorité	Séquence d'exécution
T3	4	3	EER1E
T2	8	2	EEE
T1	1	1	EER1R1R1R1E



# Ordonnancement temps réel (préemptif à priorités) de tâches dépendantes → PIP = Priority Inheritance Protocol)



# Exercice 1 : Ordonnancement (Hiv13)

Considérez un système d'exploitation monoprocesseur doté d'un ordonnanceur préemptif, à priorités (0 étant la plus faible priorité). Supposez les processus suivants :

Processus	Date d'arrivée	Séquence d'exécution	Priorité
A	2	EERE	10
B	5	EEEE	7
C	0	ERRRE	3

Les processus A et C partagent la ressource R (en exclusion mutuelle).

- Donnez le diagramme de Gantt de l'ordonnancement des processus entre les instants 0 et 13. Sur le diagramme, indiquez les accès des processus à la ressource R.**
- Y a-t-il une inversion de priorités ? Si oui, précisez les processus concernés, l'instant de début, ainsi que la durée de l'inversion de priorités. Cette durée dépend-elle du temps d'exécution du processus B ?**
- Donnez le diagramme de Gantt de l'ordonnancement des processus dans le cas où le protocole PIP est utilisé pour traiter les inversions de priorités entre les instants 0 et 13.**



# Exercice 2 : Moniteurs et variables de condition (Aut 2014)

Pour permettre à un ensemble de threads d'un même processus de partager des données de type pile, on décide d'implémenter un moniteur *pile\_t*. On vous donne le code à compléter du moniteur *pile\_t* :

```
Moniteur pile_t
{   const int N = 2; // la taille de la pile
    int P[N]; // la pile
    .... // autres variables ou constantes
    void Empiler ( int o) { .... }; // doit bloquer si la pile est pleine
    int Depiler () { ... .};// doit bloquer si la pile est vide
}
```

Complétez le moniteur pile\_t.



## Exercice 3 : Gestion de la mémoire (Hiv16)

Considérez un système monoprocesseur avec une gestion de mémoire par pagination pure (pagination à la demande) et des tables de pages à un niveau.

- La mémoire physique est composée de 4 cadres.
- La taille de chaque cadre est de 4 KiO. L'adresse virtuelle est codée sur 16 bits.
- Supposez que 2 processus P1 et P2, composés respectivement de 7 et 5 pages, arrivent dans le système, l'un à la suite de l'autre.
- Le système charge dans l'ordre, les pages 0 et 1 de P1 dans les cadres 1 et 2, et la page 1 de P2 dans le cadre 3, avant de commencer l'exécution des processus P1 et P2 (pré-pagination).
- Supposez que lorsqu'un défaut de page se produit et qu'un retrait de page est nécessaire, le système effectue un remplacement global en utilisant LRU.

a) Donnez l'adresse physique de l'adresse virtuelle : 0001 0011 0111 1000, pour

chacun des cas suivants :

- P1 référence cette adresse virtuelle et
- P2 référence cette adresse virtuelle.



## **Exercice 3 : Gestion de la mémoire (Hiv16)**

b) Représentez l'évolution de l'état de la mémoire physique, à partir de l'état courant, dans le cas où le processeur reçoit, dans l'ordre suivant, les accès aux pages des processus P1 et P2 :

**(0, P1) (1, P1) (1, P2) (5, P1) (4, P2) (5, P1) (6, P1) (1, P1) (2, P1)**  
où (0, P1), par exemple, référence la page 0 du processus P1.

Donnez aussi le nombre de défauts de pages provoqués par chaque processus.



# Exercice 4 : Windows (Hiv17)

Expliquez comment implémenter sous Windows, en utilisant l'API *win32*, le traitement réalisé par la commande Linux « *p.exe > fich* » où *p.exe* est un exécutable et *fich* est un nouveau fichier (qui n'existe pas dans le répertoire courant).

Ne donnez pas de code. Indiquez, par contre, toutes les étapes à suivre sous forme de commentaires.



# Exercice 5 : Interblocage (Hiv 2013)

Les threads `thread_p1`, `thread_p2`, et `thread_p3` d'un même processus s'exécutent en parallèle. Dites si ce processus cause nécessairement un interblocage, ne peut causer un interblocage ou peut parfois causer un interblocage. Donnez le cas échéant un ordonnancement des actions qui mène à un interblocage et un qui parvient à compléter sans interblocage. Un ordonnancement est représenté par la séquence des énoncés (chacun identifié par une lettre) exécutés.

```
void* thread_p1(void *arg) {  
    A: mutex_lock(&r2);  
    B: mutex_lock(&r3);  
    C: /* do something */  
    D: mutex_unlock(&r3);  
    E: mutex_lock(&r4);  
    F: /* do something */  
    G: mutex_unlock(&r4);  
    H: mutex_unlock(&r2);  
    return NULL ;  
}
```

```
void* thread_p2(void *arg) {  
    I: mutex_lock(&r4);  
    J: mutex_lock(&r5);  
    K: mutex_lock(&r1);  
    L: /* do something */  
    M: mutex_unlock(&r1);  
    N: mutex_unlock(&r5);  
    O: mutex_unlock(&r4);  
    return NULL ;  
}
```

```
void* thread_p3(void *arg) {  
    P: mutex_lock(&r1);  
    Q: mutex_lock(&r2);  
    R: /* do something */  
    S: mutex_unlock(&r2);  
    T: mutex_unlock(&r1);  
    return NULL ;  
}
```



## Exercice 6 : Ordonnancement (Aut15)

Supposez que les processus P1 et P2 ont chacun une section critique (le processus P3 n'a pas de section critique). Le triplet  $(x_i, y_i, z_i)$  du temps d'exécution du processus  $P_i$  (pour  $i=1,2$ ) signifie que :  $P_i$  réalise un calcul de  $x_i$  unités de temps CPU avant la demande d'entrer en section critique. Pour exécuter et quitter sa section critique,  $y_i$  unités de temps CPU sont nécessaires. Après sa section critique,  $P_i$  se termine au bout de  $z_i$  unités de temps CPU. L'ordonnancement de ces processus est circulaire avec un quantum de 4 et les temps de commutation de contexte sont supposés nuls.

Processus	Date d'arrivée ( $O_i$ )	Durée d'exécution ( $C_i$ )
P1	0	10 (3, 6, 1)
P2	1	5 (2, 2, 1)
P3	2	7

Donner le diagramme de Gant de l'exécution des processus P1, P2 et P3, pour chacun des cas suivants :

- une attente passive d'accès aux sections critiques.
- une attente active d'accès aux sections critiques.



Indiquer sur les diagrammes les dates d'entrées et de sorties des sections critiques ainsi que les attentes (passives ou actives). Donnez le temps de séjour moyen pour chaque cas.

# Exercice 7 : Gestion de la mémoire

## Caractéristiques du système :

- Mémoire physique de 64 mots de 8 bits (64 octets)
- Taille d'une page = 4 mots = 4 octets
- Table de pages à deux niveaux.
- Adresse virtuelle sur 6 bits : 2 bits (niv. 1), 2 bits (niv. 2), 2 bits (décalage)
- Une entrée dans la table de pages sur 8 bits:
  - 2 bits de contrôle : bit de présence, bit de référence (10,11)  
si bit de présence =0 et bit de référence =1, la page est dans la zone de « swap »
  - 6 bits pour l'adresse
- Deux processus P1 et P2 sont en mémoire.
- La table de pages du premier niveau de P1 commence à 32
- La table de pages du premier niveau de P2 commence à 52.
- L'état courant de la mémoire physique et de la zone de swap :



# Exercice 7 : Gestion de la mémoire

État de la mémoire physique (16 cadres de 4 octets chacun)

0	1	1	1
1	1	1	2
2	0	0	0
3	0	0	0
4	1	0	6
5	0	1	9
6	0	0	59
7	1	0	62
8	1	0	7
9	1	0	6
10	1	0	5
11	0	1	2
12	0	0	0
13	0	0	30
14	0	0	0
15	0	0	0

16	0	0	0
17	0	0	0
18	0	0	0
19	0	0	0
20	1	1	4
21	1	1	44
22	0	1	3
23	1	1	12
24	0	0	0
25	0	0	0
26	0	1	0
27	1	0	16
28			
29			
30			
31			

32	0	0	4
33	1	1	36
34	1	1	20
35	0	0	52
36	0	1	2
37	0	0	0
38	0	0	0
39	0	0	0
40			
41			
42			
43			
44	1	0	48
45	1	0	52
46	0	0	56
47	0	0	60

48	1	0	8
49	1	0	0
50	0	0	0
51	0	0	0
52	1	1	48
53	1	1	24
54	0	0	28
55	0	0	12
56	1	0	8
57	1	1	7
58	1	0	6
59	1	0	5
60	1	0	0
61	1	0	56
62	0	0	0
63	0	0	0

Pages déplacées en mémoire secondaire (va-et-vient)

Page 0

0	0	40
0	0	0
0	0	0
0	0	0

Page 1

0	1	10
0	0	12
0	0	14
1	0	16

Page 2

0	1	1
0	0	0
0	0	0
1	0	0

Page 3

1	0	63
0	0	29
0	1	0
0	0	0



## Exercice 7 : Gestion de la mémoire

a) Pour chaque page du processus P1, indiquez laquelle des situations suivantes s'applique (utilisez le tableau à la dernière page):

- Présente en mémoire (spécifiez dans quel cadre)
- Absente de la mémoire ou invalide
- Dans le va-et-vient (indiquez à quelle position)

b) Pour chacun des 16 cadres de la mémoire, indiquez laquelle des situations suivantes s'applique (utilisez le tableau à la dernière page):

1. Contient une page d'un processus (indiquez le processus)
2. Contient une table de pages (indiquez le processus)
3. Libre
4. Autre

c) Le processus P1 veut accéder aux adresses logiques 37 et 40? Dans chaque cas, indiquez s'il y aura faute de page, l'adresse physique obtenue, ainsi que le contenu de l'octet à cette adresse. *Remarque:* s'il y a faute de page, indiquez les changements à la mémoire, une fois complété le traitement de cette faute de page.

# Exercice 7 : Gestion de la mémoire (indication)

Table de pages de P1

32

0	0	4
1	1	36
1	1	20
0	0	52

36

0	1	2
0	0	0
0	0	0
0	0	0

Table de pages de P2

52

1	1	48
1	1	24
0	0	28
0	0	12

20

1	1	4
1	1	44
0	1	3
1	1	12

48

1	0	8
1	0	0
0	0	0
0	0	0



## Exercice 7 : Gestion de la mémoire

d) Supposons que le processus P1 ne peut occuper plus de trois cadres en mémoire et que l'algorithme de l'horloge est utilisé pour le remplacement de page. Supposons aussi que le système n'utilise aucun autre algorithme de remplacement global. Voici des exemples de séquences d'accès à des pages réalisées par le processus P1 depuis le début de son exécution jusqu'au moment où la mémoire se retrouve dans l'état illustré à la figure 1:

- 4, 10, 8, 9, 11, 8
- 8, 9, 11, 8, 9
- 10, 9, 8, 11

Indiquez lesquelles, parmi ces séquences, peuvent mener à l'état de la mémoire illustré à la figure 1? (Justifiez votre réponse)



# Exercice 8 : Ordonnancement (Aut15)

Supposez trois processus P1, P2 et P3 avec les caractéristiques suivantes :

Processus	Date d'arrivée (O <sub>i</sub> )	Durée d'exécution (C <sub>i</sub> )
P1	0	10
P2	1	5
P3	2	7

Donnez le diagramme de Gant de l'exécution de ces processus dans le cas d'un ordonnancement circulaire de quantum de 4. Donnez les temps de séjour et d'attente moyens (TMS et TMA). Les temps de commutation de contexte sont supposés nuls.



## Exercice 9 : Gestion de la mémoire (Hiv14/Aut14)

Un système qui implémente la pagination à la demande dispose de 4 cadres (cases) de mémoire physique qui sont toutes occupées, à un instant donné. La tableau suivant indique, pour chaque case de mémoire physique, la date de chargement de la page qu'elle contient ( $t_{\text{chargement}}$ ), la date du dernier accès à cette page ( $t_{\text{dernier-accès}}$ ) et les bits de modification (M) et de présence (P). Les dates sont données en tops d'horloge.

Case	$t_{\text{chargement}}$	$t_{\text{dernier-accès}}$	M	P
0	126	270	0	1
1	230	255	0	1
2	110	260	1	1
3	180	275	1	1

- Indiquez la page qui sera remplacée en cas de défaut de page, pour chacun des algorithmes de remplacement de pages suivants : 1) LRU et 2) FIFO
- Supposez qu'un système paginé réserve 4 cadres physiques libres à chaque processus créé. Aucun autre cadre n'est alloué au processus. Durant son exécution, un processus référence dans l'ordre les pages : 0 1 7 2 3 2 7 1 0 3. Donnez pour chacun des algorithmes de remplacement de pages suivants, l'évolution de l'état des cadres 0, 1, 2 et 3 réservés au processus ainsi que le nombre de défauts de pages générés : FIFO et LRU.



# Exercice 9 : Windows (Hiv14)

Vous devez implémenter une queue bloquante, de dimension maximale fixe, qui peut accepter des requêtes de plusieurs fils (threads) d'exécution, en utilisant l'API de Windows (par exemple pour les primitives de synchronisation comme les sémaphores et les mutex). Fournissez la déclaration des champs de données de la classe Queue et fournissez une implémentation pour son constructeur, son destructeur et ses méthodes enqueue et dequeue dont la signature est fournie. Pour simplifier le problème, vous n'avez pas besoin de vérifier les valeurs de retour pour les conditions d'erreur.

```
Queue::Queue(intcapacity);  
Queue::~Queue();  
void Queue::enqueue(void *item);  
void *Queue::dequeue();
```



# Exercice 10 : Moniteurs et variables de condition (Hiv17)

Considérez le moniteur ProducteurConsommateur suivant (cas d'un producteur et d'un consommateur) :

*Moniteur ProducteurConsommateur*

```
{ const N=100 ;
    int tampon[N], compteur =0, ic=0, ip=0 ;
    boolc nplein, nvide ; // variables de condition
    void produire (int pid, int objet) // pid numéro du producteur
    { if (compteur==N) wait(nplein) ;
        tampon[ip] = objet ;
        ip = (ip+1)%N ; compteur++ ;
        if (compteur==1) signal(nvide) ;
    }
    int consommer (int cid ) // cid numéro du consommateur
    { int objet ;
        if (compteur ==0) wait(nvide) ;
        objet = tampon[ic] ;
        ic = (ic+1)%N ; compteur -- ;
        if(compteur==N-1) signal(nplein) ;
        return objet ;
    }
}
```



# Exercice 10 : Moniteurs et variables de condition (Hiv17)

Supposez qu'il y a exactement 3 producteurs (numérotés de 0 à 2) et 3 consommateurs (numérotés de 0 à 2).

On veut que :

- les producteurs produisent tour à tour (producteur 0, producteur 1, producteur 2, producteur 0, etc.) et que
- les consommateurs consomment aussi tour à tour (consommateur 0, consommateur 1, consommateur 2, consommateur 0, etc.).

Modifiez le pseudo-code précédent de manière à satisfaire les requis précédents (productions tour à tour et consommations tour à tour). Pour la synchronisation, vous devez vous limiter à l'utilisation de variables de condition dans le moniteur.



# Exercice 11 (RMA + PIP)

Tâche	Date d'arrivée	Temps d'exécution Ci	Période Pi
T3	3	3: ER1E	5
T2	4	2: EE	10
T1	1	5 : ER1R1R1E	20

- Les tâches T1, T2 et T3 sont-elles ordonnableables, selon RMA ?
- A-t-on un problème d'inversion de priorité ? Si oui, donnez le diagramme de Gantt correspondant au cas où ce problème est traité en utilisant le protocole PIP (Priority Inheritance Protocol). A-t-on des échéances manquées, dans ce cas ?
- Mission Mars Pathfinder lancée par la NASA (décembre 1996)  
<http://www.fil.univ-lille1.fr/~nebut/portail/svl/fichiers/tp/tpPathfinder/tpPathfinder.pdf>



Tâche météo (la moins prioritaire)

Tâche gestion du bus la plus prioritaire

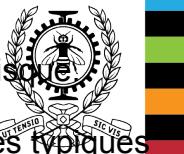
Tâche de communication (priorité intermédiaire)

([http://degeeter.pagesperso-orange.fr/inv\\_fr.htm](http://degeeter.pagesperso-orange.fr/inv_fr.htm))

# La suite

( <http://www.zdnet.fr/actualites/chiffres-cles-les-systemes-d-exploitation-sur-pc-39790131.htm> )

- NF3610 systèmes embarqués (SE temps réel) Introduction aux systèmes embarqués. Modélisation d'un système embarqué. Architecture d'une plate-forme pour systèmes embarqués : processeur embarqué, système multibus, accélérateurs matériels (coprocesseurs) et mémoires. Interface entre la partie matérielle et la partie logicielle d'un système embarqué. Conception de la partie logicielle d'un système embarqué. **Système d'exploitation temps réel. Types d'ordonnancement. Analyse du temps de réponse.** Étapes de conception d'un système embarqué.
- INF3500 Conception et réalisation de systèmes numériques Principes de base des systèmes numériques. Description de circuits numériques grâce à une combinaison de schémas : code dans un langage de description matérielle (VHDL) et diagrammes d'états. Simulation de circuits numériques. Principaux dispositifs de logique programmable : mémoires mortes (ROM), réseaux logiques programmables (PLA et PAL), circuits logiques programmables complexes (CPLD) et réseaux pré-diffusés programmables (FPGA). Technologies de programmation et planchettes de développement. Caractéristiques des FPGA. Flot de conception : description, synthèse, placement, routage et programmation. Notions avancées de design pour FPGA. Exemples d'application.
- INF8601 Systèmes informatiques parallèles Taxonomie et organisation des systèmes informatiques parallèles. Architectures avancées de **multiprocesseurs**. Hiérarchie de mémoires, **protocoles de cohérence des antémémoires**. **Parallélisme par fils d'exécution multiples**. Conception **d'applications parallèles en mémoire partagée**. Coprocesseurs pour le calcul parallèle. Grappes de calcul et échange de messages entre les noeuds. **Techniques d'équilibrage de charge**. Infrastructure. Conception d'applications parallèles en mémoire répartie.
- INF3405 Réseaux informatiques Classification des réseaux. Techniques de commutation. Architectures technologiques de transmission. Tramage, détection d'erreurs, contrôle du flot et contrôle d'erreurs par retransmission. Architecture des réseaux : modèle par couches, relations entre les couches et primitives de contrôle. Protocoles des réseaux locaux : Ethernet et réseaux sans fil. Architecture technologique TCP/IP (Transport Control Protocol/Internet Protocol) : modèle, adressage, protocoles et routage. Analyse de la qualité de service et modèles pour les réseaux informatiques. Mécanismes améliorant la qualité de service. IP version 6 et passage à la version 6. Contrôle et analyse de la congestion avec TCP. Applications de TCP/IP.
- INF4420A Sécurité informatique Définition, portée et objectifs de la sécurité informatique. Méthodologie d'analyse et de gestion du risque. Éléments de cryptographie et de cryptanalyse. Algorithmes de chiffrement à clé privée et à clé publique. Fonctions de hachage cryptographique. Signatures numériques. Gestion des clés et infrastructures à clés publiques. Sécurité des logiciels. Vulnérabilités typiques et techniques d'exploitation. Logiciels malicieux et contre-mesures. Sécurité des systèmes d'exploitation. Mécanismes d'authentification, contrôle d'accès et protection de l'intégrité. Modèles de gestion du contrôle d'accès. Sécurité des bases de données et des applications Web. Sécurité des réseaux. Configuration sécuritaire. Coupe-feux, détecteurs d'intrusions et serveur mandataire. Protocoles de réseaux sécurisés. Organisation et gestion de la sécurité informatique. Acteurs et types d'interventions. Normalisation et organismes pertinents. Cadre légal et déontologique.



# Exercice 1 : Ordonnancement (Hiv13) - Solution

b) i)

C	C	A	A	C	B	B	B	B	C	A	A	C
0	1	2	3	4	5	6	7	8	9	10	11	12
R		R			R		R		R	R		

ii) Oui. À l'instant 4, on a une inversion de priorités de 6 unités de temps. Le processus A, qui possède la priorité la plus élevée, est empêché de s'exécuter par le processus C, qui accède à la ressource R entre les instants 4 et 5 puis entre 9 et 10, et par le processus B, qui suspend le processus C entre les instants 5 et 9. Oui, la durée de l'inversion de priorités dépend du temps d'exécution de B, puisque C est préempté par B (qui est plus prioritaire). C ne pourra pas s'exécuter avant que B se termine ou se bloque.

iii)

C	C	A	A	C	C	A	A	B	B	B	B	C
0	1	2	3	4	5	6	7	8	9	10	11	12
R		R	R	R		R		R		R	R	



# Exercice 2 : Moniteurs et variables de condition (Aut 2014) - Solution

## Moniteur pile\_t

```
{ const int N = 2; // la taille de la pile
  int P[N]; // la pile
  int s = 0; // le sommet de la pile
  boolc nplein,, nvide;
  // pour bloquer si la pile est vide (pleine)
  void Empiler ( int o ) {
    while(s==N) wait(nplein); P[s] = o; s++; if(s==1) signal(nvide);
  };
  int Depiler( )
  { int t;
  while(s==0) wait(nvide); s--; t=P[s] ; if(s==N-1) signal(nplein); return t;};
}
```



## Exercice 3 : Gestion de la mémoire (Hiv16)

a) La taille d'une page est 4kiO. Le déplacement est donc codé sur 12 bits.

L'adresse virtuelle est sur 16 bits : 4 bits pour le numéro de page et 12 bits pour le déplacement dans la page.

P1 : 00 01 00 11 01 11 10 00 => il s'agit de la page 1 de P1. Cette page est chargée dans le cadre 2 => L'adresse physique est : 00 10 00 11 01 11 10 00

P2 : 00 01 00 11 01 11 10 00 => il s'agit de la page 1 de P2. Cette page est chargée dans le cadre 3 => L'adresse physique est : 00 11 00 11 01 11 10 00

b) LRU : 5 défauts de pages au total (4 par P1 et 1 par P2)

Etat courant	(0, P1)	(1, P1)	(1, P2)	(5, P1)	(4, P2)	(5, P1)	(6, P1)	(1, P1)	(2, P1)
				(5,P1)	(5,P1)	(5,P1)	(5,P1)	(5,P1)	(5,P1)
(0,P1)	(0,P1)	(0,P1)	(0,P1)	(0,P1)	(4,P2)	(4,P2)	(4,P2)	(4,P2)	(2,P1)
(1,P1)	(1,P1)	(1,P1)	(1,P1)	(1,P1)	(1,P1)	(1,P1)	(6,P1)	(6,P1)	(6,P1)
(1,P2)	(1,P2)	(1,P2)	(1,P2)	(1,P2)	(1,P2)	(1,P2)	(1,P2)	(1,P1)	(1,P1)



# Exercice 4 : Windows (Hiv17)

Expliquez comment implémenter sous Windows, en utilisant l'API *win32*, le traitement réalisé par la commande Linux « *p.exe > fich* » où *p.exe* est un exécutable et *fich* est un nouveau fichier (qui n'existe pas dans le répertoire courant). Ne donnez pas de code.  
Indiquez, par contre, toutes les étapes à suivre sous forme de commentaires clairs et précis.

**Il suffit de:**

- 1) créer/ouvrir le fichier Fich (CreateFile), qui retourne le handle du fichier créé ou ouvert.**
- 2) rendre héritable le fichier handle du fichier.**
- 3) déclarer deux variables pi de type PROCESS\_INFORMATION et si de type STARTUPINFO; Initialiser les attributs hStdInput à STDIN du processus, hStdError à STDERR du processus et hStdOutput au handle du fichier Fich créé.**
- 4) créer un processus fils : CreateProcess("p.exe", //name or path of executable  
NULL, // no command line.  
NULL, // Process handle not inheritable.  
NULL, // Thread handle not inheritable.  
true, // Set handle inheritance to true.  
0, // No creation flags.  
NULL, // Use parent's environment block.  
NULL, // Use parent's starting directory.  
&si, // Pointer to STARTUPINFO structure.  
&pi )**
- 5) fermer le handle du fichier.**
- 6) attendre la fin du fils.**



# Exercice 5 : Interblocage (Hiv 2013) et sol

Les threads `thread_p1`, `thread_p2`, et `thread_p3` d'un même processus s'exécutent en parallèle. Dites si ce processus cause nécessairement un interblocage, ne peut causer un interblocage ou peut parfois causer un interblocage. Donnez le cas échéant un ordonnancement des actions qui mène à un interblocage et un qui parvient à compléter sans interblocage. Un ordonnancement est représenté par la séquence des énoncés (chacun identifié par une lettre) exécutés.

```
void* thread_p1(void *arg) {  
    A: mutex_lock(&r2);  
    B: mutex_lock(&r3);  
    C: /* do something */  
    D: mutex_unlock(&r3);  
    E: mutex_lock(&r4);  
    F: /* do something */  
    G: mutex_unlock(&r4);  
    H: mutex_unlock(&r2);  
    return NULL ;  
}
```

```
void* thread_p2(void *arg) {  
    I: mutex_lock(&r4);  
    J: mutex_lock(&r5);  
    K: mutex_lock(&r1);  
    L: /* do something */  
    M: mutex_unlock(&r1);  
    N: mutex_unlock(&r5);  
    O: mutex_unlock(&r4);  
    return NULL ;  
}
```

```
void* thread_p3(void *arg) {  
    P: mutex_lock(&r1);  
    Q: mutex_lock(&r2);  
    R: /* do something */  
    S: mutex_unlock(&r2);  
    T: mutex_unlock(&r1);  
    return NULL ;  
}
```

Il existe beaucoup d'ordonnancements qui ne causeront pas d'interblocage, comme tout `thread_p1`, suivi de tout `thread_p2` et de tout `thread_p3`, soit A B C D E F G H... T. Un interblocage se produirait, si on a la séquence A B C D I J P car `thread_p1` essaiera d'acquérir r4 déjà pris par `thread_p2` qui lui voudra r1 déjà pris par `thread_p3` qui lui voudra r2 déjà pris par `thread_p1`. Cet ensemble peut donc parfois causer un interblocage



## Exercice 6 : Ordonnancement (Aut15) - Solution

**0 P1 3 P1 sc 4 P2 6 P3 10 P1 sc 14 P3 17 P1 sc 18 P1 19 P2 sc 21 P2 22**  
**P2 bloque à 6, il est débloqué à 18.**  $TMS = (19 + (22-1) + (17-2)) / 3 = 18,33$

**0 P1 3 P1 sc 4 P2 6 P2 aa 8 P3 12 P1 sc 16 P2 aa 20 P3 23 P1 sc 24 P1 25 P2 sc 27 P2 28**  
**(aa pour attente active).**  $TMS = (25 + (28 -1) + (23-2)) / 3 = 24,33$



# Exercice 7 : Gestion de la mémoire (indication)

Table de pages de P1

32

0	0	4
1	1	36
1	1	20
0	0	52

36

0	1	2
0	0	0
0	0	0
0	0	0

Table de pages de P2

52

1	1	48
1	1	24
0	0	28
0	0	12

20

1	1	4
1	1	44
0	1	3
1	1	12

48

1	0	8
1	0	0
0	0	0
0	0	0



# Exercice 7 : Gestion de la mémoire – Solution

a)

15	Absente/Invalide
14	Absente/Invalide
13	Absente/Invalide
12	Absente/Invalide
11	En mémoire - cadre 3
10	Zone de swap – page 3
9	En mémoire – cadre 11
8	En mémoire – cadre 1
7	Absente/Invalide
6	Absente/Invalide
5	Absente/Invalide
4	Zone de swap – page 2
3	Absente/Invalide
2	Absente/Invalide
1	Absente/Invalide
0	Absente/Invalide
#page de P1	localisation

b)

15	autre
14	autre
13	TDP de P2
12	TDP de P2
11	Page 9 de P1
10	libre
9	TDP de P1
8	TDP de P1
7	libre
6	TDP de P2
5	TDP de P1
4	Page 7 de P2
3	Page 11 de P1
2	Page 0 de P2
1	Page 8 de P1
0	Page 1 de P2
#cadre	contenu



c)

## Exercice 7 : Gestion de la mémoire – Solution

Adresse virtuelle sur 6 bits : 2 bits (1<sup>er</sup> niveau de TDP) + 2 bits (2<sup>ième</sup> niveau de TDP) + 2 bits depl.  
 $37 = 32 + 4 + 1 \rightarrow$  code binaire : 100101

Table de pages de P1

32

0	0	4
1	1	36
1	1	20
0	0	52

20

1	1	4
1	1	44
0	1	3
1	1	12

36

0	1	2
0	0	0
0	0	0
0	0	0

Adresse physique de 37 est :  $44 + 1 = 45$   
 L'octet 45 contient : 1 0 52



c)

## Exercice 7 : Gestion de la mémoire – Solution

Adresse virtuelle sur 6 bits : 2 bits (1<sup>er</sup> niveau de TDP) + 2 bits (2<sup>ième</sup> niveau de TDP) + 2 bits depl.  
 $40 = 32 + 8 \rightarrow$  code binaire : 101000

Table de pages de P1

32

0	0	4
1	1	36
1	1	20
0	0	52

20

1	1	4
1	1	44
0	1	3
1	1	12

36

0	1	2
0	0	0
0	0	0
0	0	0

La page 10 est dans la zone de swap (la 3<sup>ième</sup> page) -> un défaut de page.



- 1) Allouer un cadre libre à P1
- 2) Charger la page 3 de la zone de swap dans un cadre libre (par exemple le cadre 7).
- 3) Mettre à jour la table de pages (remplacer l'entrée (0 1 3) par (1 1 28)) .

d)

## Exercice 7 : Gestion de la mémoire – Solution

La première séquence car les pages 4 et 10 de P1 sont dans la zone de swap (ce qui signifie qu'elles ont été référencées par P1). Un scénario possible serait :

- 4 est chargée en mémoire;
- 10 est chargée en mémoire;
- 8 est chargée en mémoire;
- 9 remplace 4
- 11 remplace 10
- 8 est déjà en mémoire.

Il n'y a pas de référence à la page 4 dans les deux autres séquences. Ce qui est en contradiction avec le fait que 4 est dans la zone de swap.



# Exercice 8 : Ordonnancement (Aut15) - Solution

Supposez trois processus P1, P2 et P3 avec les caractéristiques suivantes :

Processus	Date d'arrivée (O <sub>i</sub> )	Durée d'exécution (C <sub>i</sub> )
P1	0	10
P2	1	5
P3	2	7

Donnez le diagramme de Gant de l'exécution de ces processus dans le cas d'un ordonnancement circulaire de quantum de 4. Donnez les temps de séjour et d'attente moyens (TMS et TMA). Les temps de commutation de contexte sont supposés nuls.

**0 P1 4 P2 8 P3 12 P1 16 P2 17 P3 20 P1 22**

$$\text{TMA} = ((22-10) + (16-5) + (18-7)) / 3 = 11.33$$

$$\text{TMS} = ((22) + (17-1) + (20-2)) / 3 = 18.66$$



# Exercice 9 : Gestion de la mémoire (Hiv14) - Solution

a)

- 1) Pour LRU, on retire la page la moins récemment utilisée. Il s'agit donc de choisir une page selon le critère de la colonne  $t_{\text{dernier-accès}}$ . La page à retirer est celle chargée dans la case 1, qui a été accédée pour la dernière fois à la date 255.
- 2) Pour FIFO, on retire la page qui a le plus grand temps de séjour. Il s'agit donc de suivre le critère de la colonne  $t_{\text{chargement}}$ . La page à retirer est celle chargée dans la case 2 qui est en mémoire depuis la date 110.

b)

FIFO	0	1	7	2	3	2	7	1	0	3
0	0	0	0	0	3	3	3	3	3	3
1		1	1	1	1	1	1	1	0	0
2			7	7	7	7	7	7	7	7
3				2	2	2	2	2	2	2

FIFO 6 défauts de pages

LRU	0	1	7	2	3	2	7	1	0	3
0	0	0	0	0	3	3	3	3	0	0
1		1	1	1	1	1	1	1	1	1
2			7	7	7	7	7	7	7	7
3				2	2	2	2	2	2	3

LRU 7 défauts de pages.



# Exercice 9 : Windows (Hiv14)

## Solution

```
void **queue;  
HANDLE sem_free, sem_busy, mutex;  
int size, ip, ic;  
  
Queue::Queue(int capacity) {  
    size = capacity;  
    queue = new void*[size];  
    ip = ic = 0;  
    sem_free = CreateSemaphore(NULL, size, size, NULL);  
    sem_busy = CreateSemaphore(NULL, 0, size, NULL);  
    mutex = CreateMutex(NULL, FALSE, NULL);  
}  
}
```

```
void Queue::enqueue(void *item) {  
    WaitForSingleObject(sem_free, INFINITE);  
    WaitForSingleObject(mutex, INFINITE);  
    queue[ip] = item;  
    ip = (ip + 1) % size;  
    ReleaseMutex(mutex);  
    ReleaseSemaphore(sem_busy, 1, NULL);  
}
```

```
void *Queue::dequeue() {  
    void *item;  
    WaitForSingleObject(sem_busy, INFINITE);  
    WaitForSingleObject(mutex, INFINITE);  
    item = queue[ic];  
    ic = (ic + 1) % size;  
    ReleaseMutex(mutex);  
    ReleaseSemaphore(sem_free, 1, NULL);  
    return item;  
}
```



# Exercice 10 : Moniteurs et variables de condition (Hiv17)

Moniteur ProducteurConsommateur

```
{ const N=100 ;
int tourp =0, tourc=0 ;
int tampon[N], compteur =0, ic=0, ip=0 ;
boolc nplein, nvide ; // variables de condition
boolc wtourp[3], wtourc[3] ;
void produire (int pid, int objet)
{ if (tourp !=pid) wait(wtourp[pid]) ;
  if (compteur==N) wait(nplein) ;
  tampon[ip] = objet ;
  ip = (ip+1)%N ; compteur++ ;
  if (compteur==1) signal(nvide) ;
  tourp = (tourp+1) % 3 ;
  signal(wtourp[tourp]) ;
}
}
```

```
int consommer (int cid )
{ int objet ;
  if (tourp !=cid) wait(wtourc[cid]) ;
  if (compteur ==0) wait(nvide) ;
  objet = tampon[ic] ;
  ic = (ic+1)%N ; compteur -- ;
  if(compteur==N-1) signal(nplein) ;
  tourc = (tourc+1) % 3 ;
  signal(wtourc[tourc]) ;
  return objet ;
}
```

