

Noyau d'un système d'exploitation

INF2610

Séances d'exercices



Département de génie informatique et génie logiciel

POLYTECHNIQUE
MONTRÉAL



AFFILIÉE À
L'UNIVERSITÉ DE MONTRÉAL



Séance d'exercices

Gestion de la mémoire

Problèmes classiques de synchronisation (Pb. des philosophes, Pb. des lecteurs et rédacteurs)

Moniteurs & Interblocage

Windows

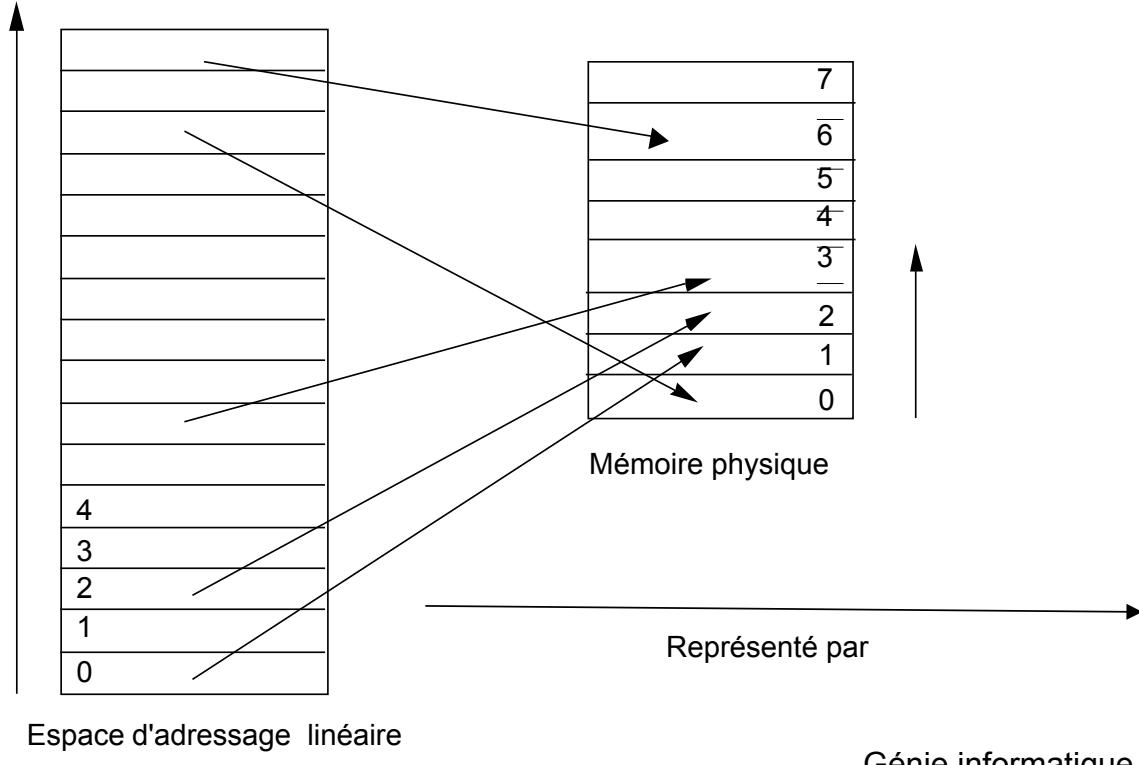
Ordonnancement classique et ordonnancement temps réel



Gestion de la mémoire

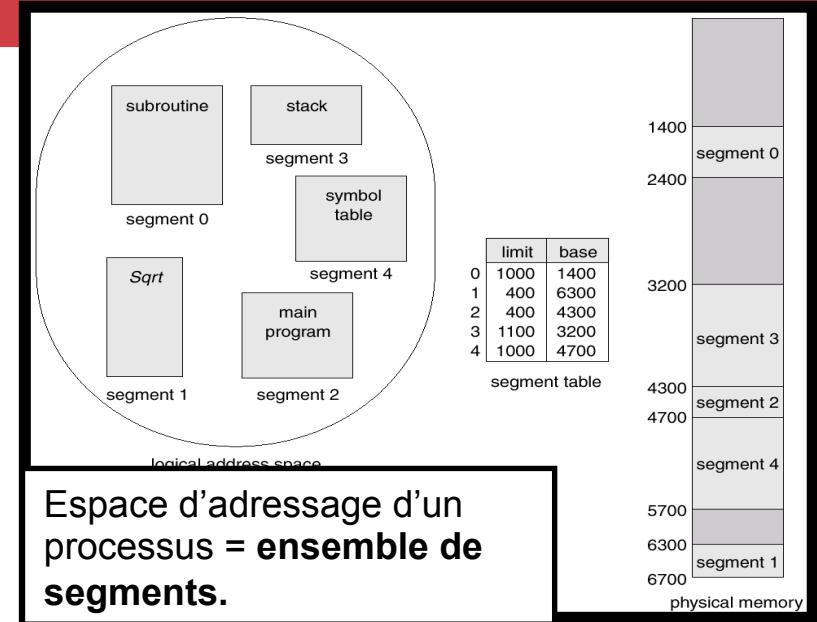
- Chaque processus a son propre espace d'adressage et accède à la mémoire physique via cet espace.

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



Espace d'adressage linéaire

Noyau d'un système d'exploitation



1	6
0	
1	0
0	
0	
0	
1	3
0	
4	0
3	0
2	1
1	2
1	0
0	1

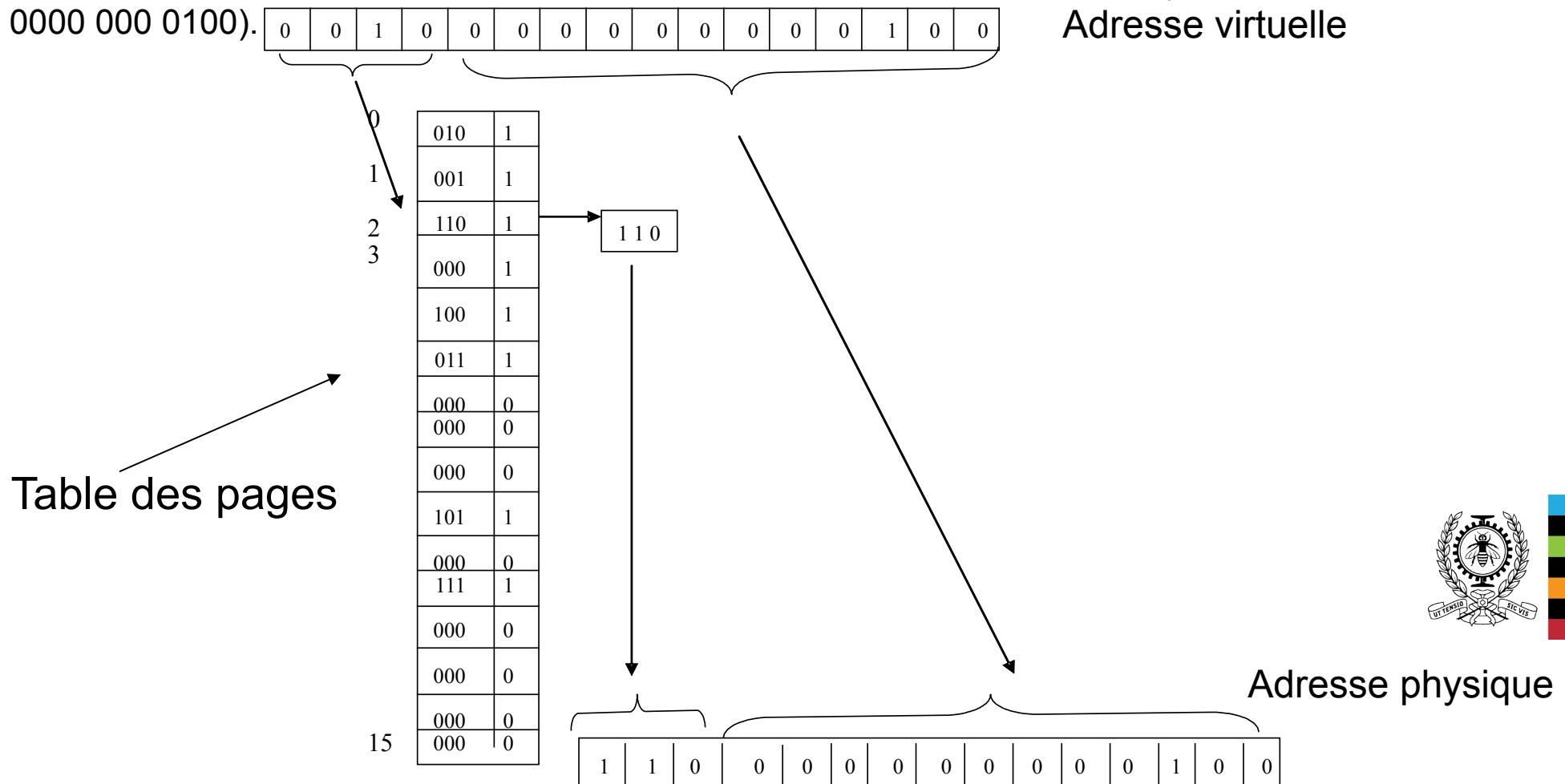
Table des pages

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

Gestion de la mémoire

Conversion d'adresses virtuelles (pagination pure)

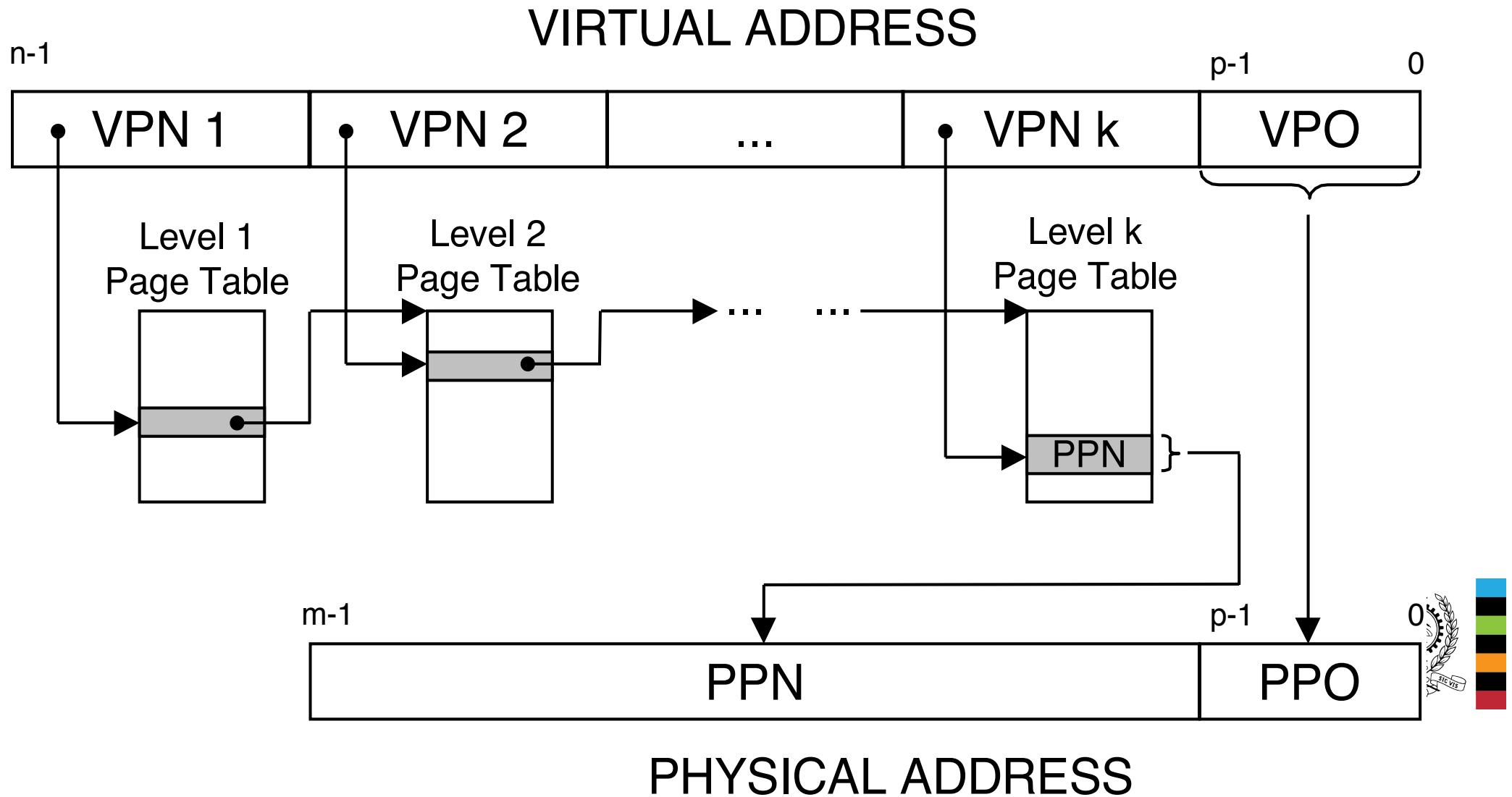
- Adresse virtuelle = (numéro de page, déplacement dans la page).
- Chaque entrée de la table des pages est composée de plusieurs champs, notamment : bit de présence (P), bit de référence (R), bits de protection (un, deux ou trois bits), bit de modification (M appelé bit dirty/clean), numéro de case 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

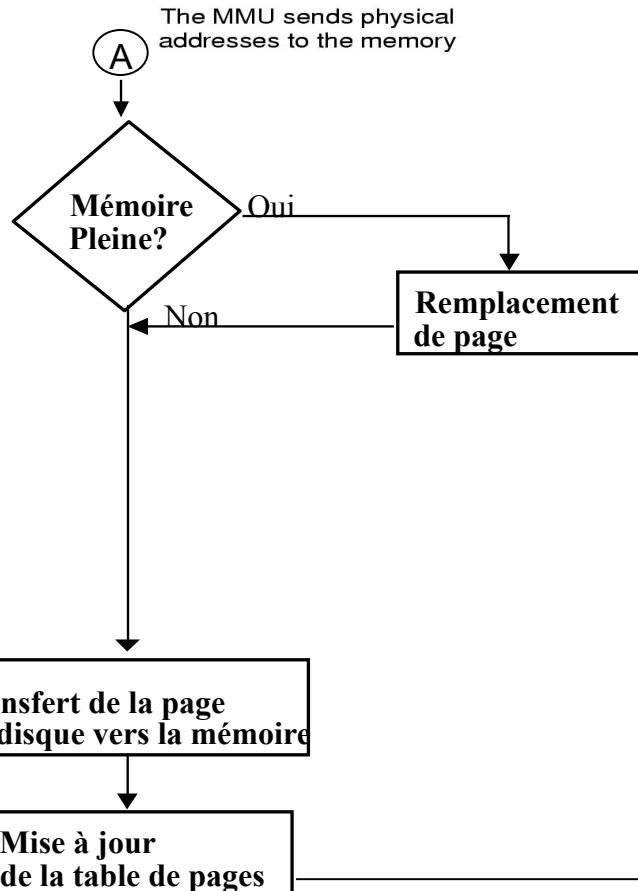
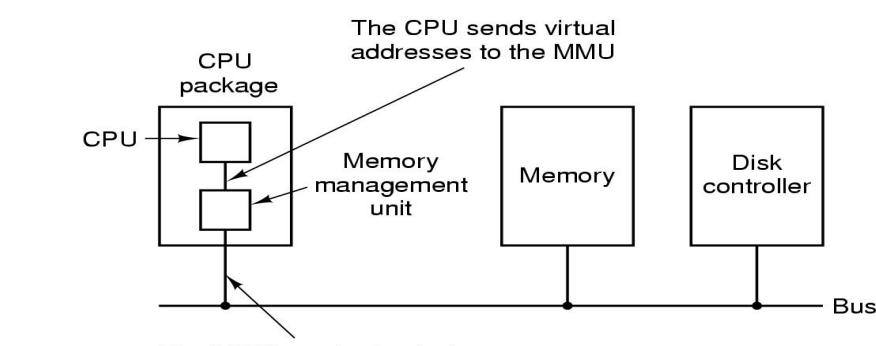
Conversion d'adresses virtuelles (pagination pure)

Table de pages à plusieurs niveaux

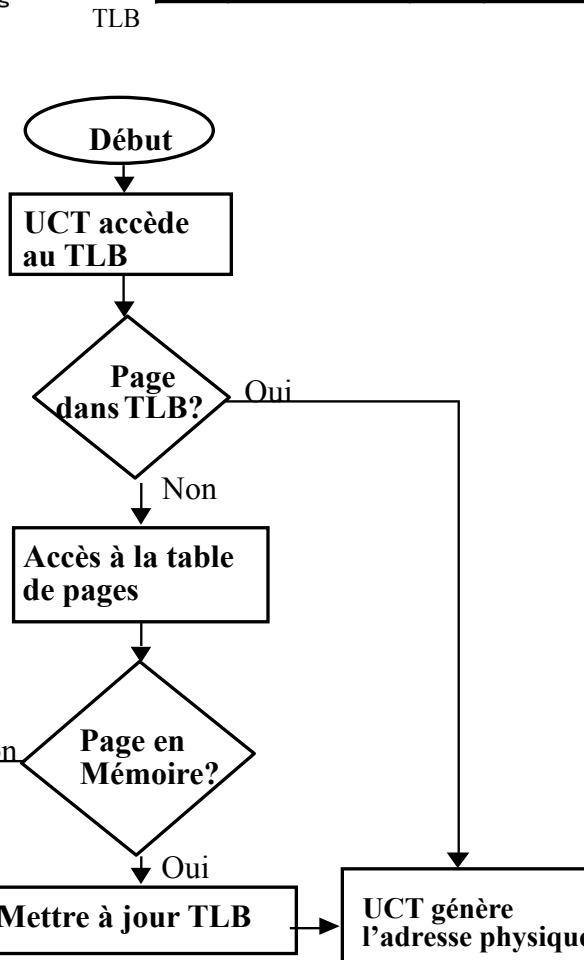


Gestion de la mémoire

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

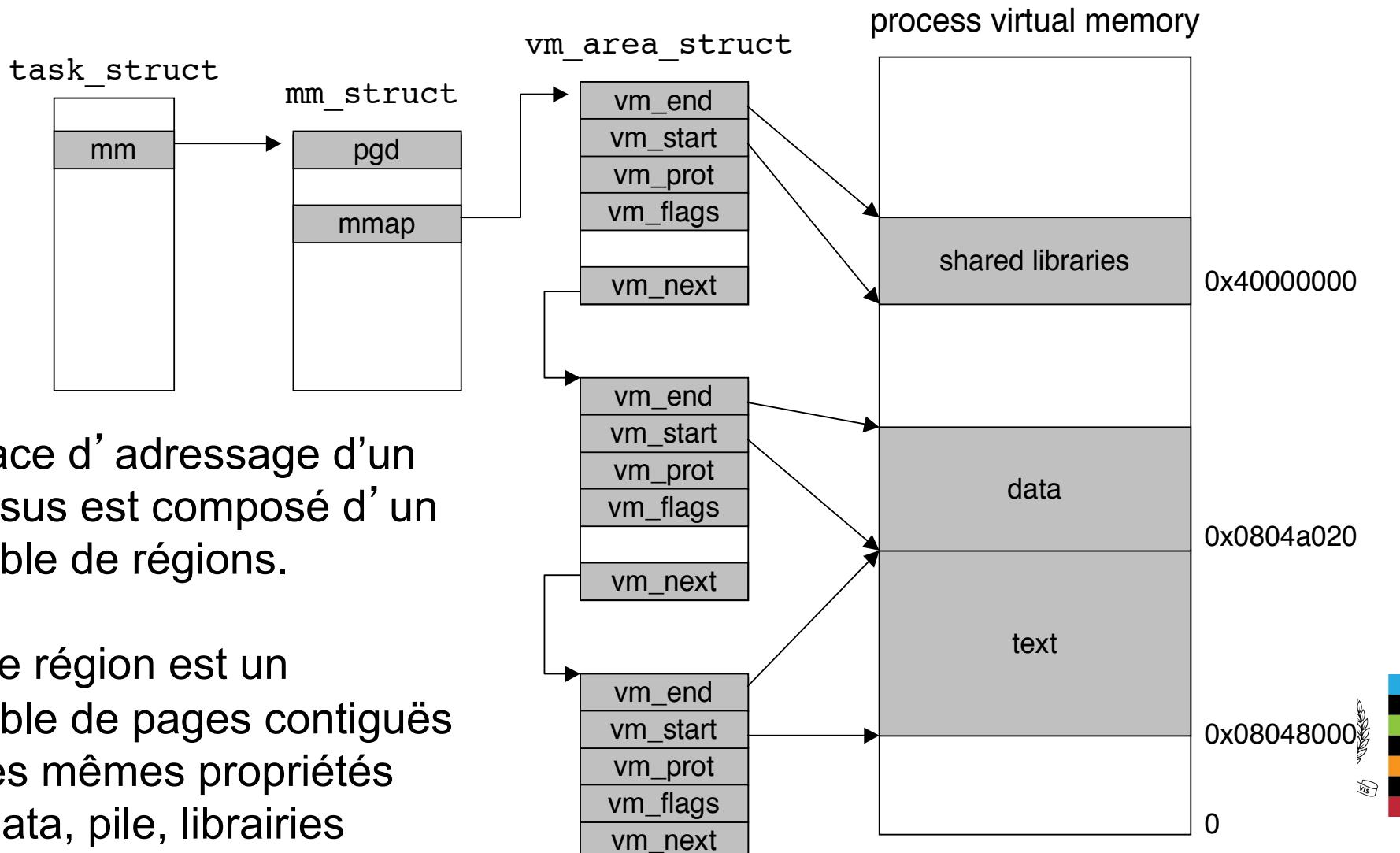


- Algorithmes de remplacement de page mémorisent les références passées aux pages.
- Le choix de la page à retirer dépend des références passées : LRU, Horloge, FIFO, Aléatoire (et Optimal).
- En cas de défaut de page, chargement de plusieurs pages voisines en mémoire.
- Allocation effective est retardée jusqu'à la référence ou l'accès en écriture.

Gestion de la mémoire : Cas de Linux

Espace d'adressage d'un processus

- 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.).

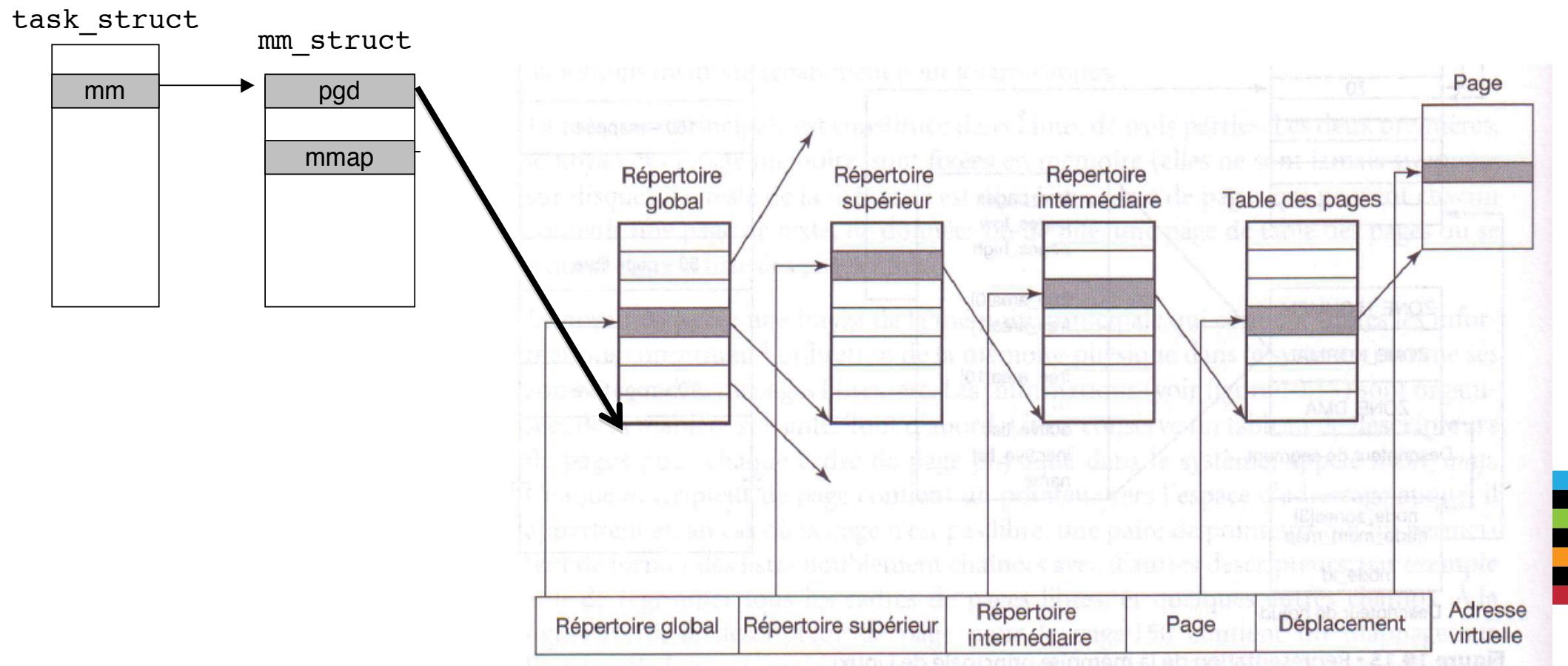


Gestion de la mémoire : Cas de Linux

Espace d'adressage d'un processus

Adresse virtuelle

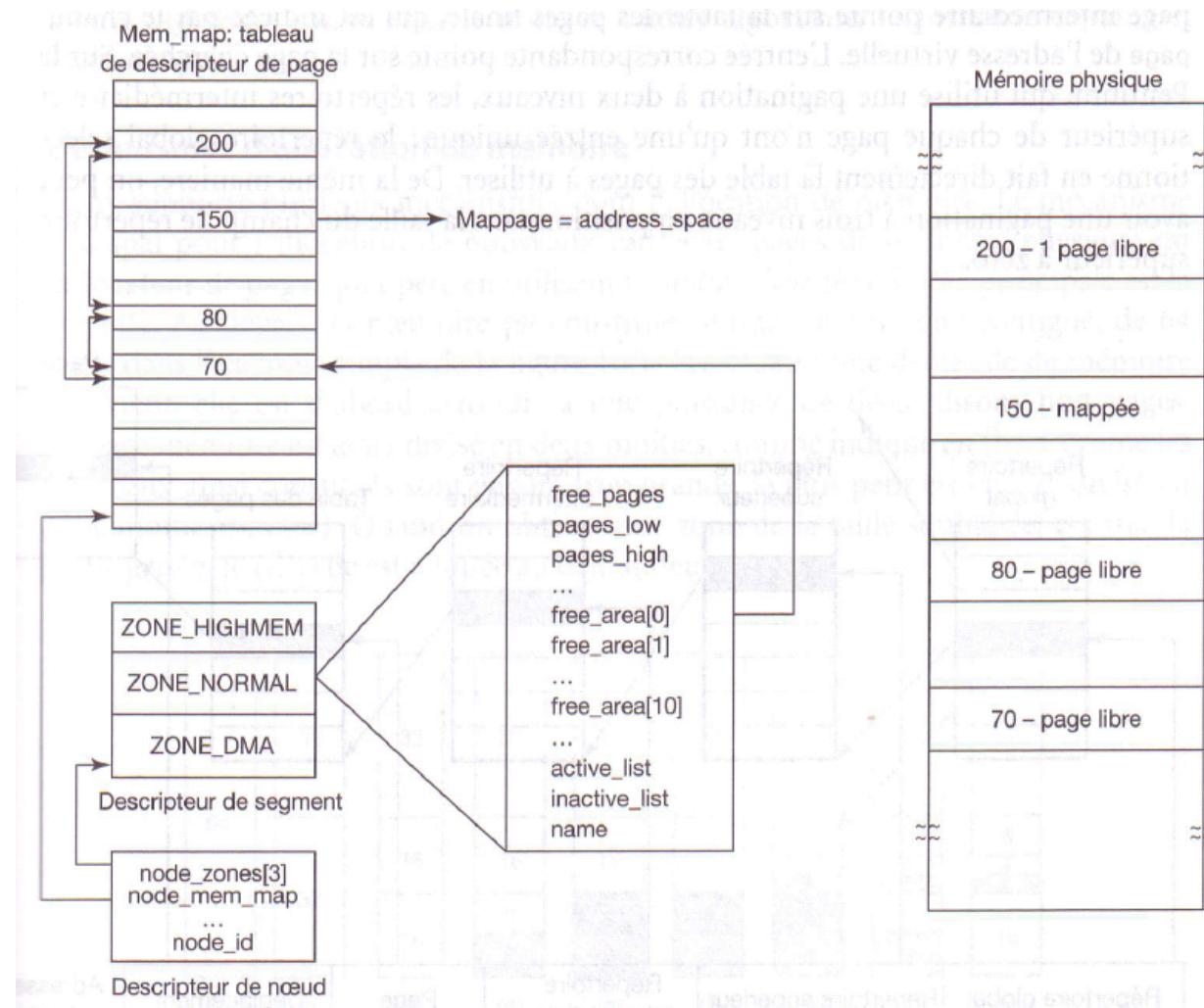
Table des pages à 4 niveaux



Gestion de la mémoire : Cas de Linux

Mémoire physique

Structures de données associées (vue d'ensemble)



- L'algorithme de base d'allocation d'espace physique est un allocateur par subdivision.



Problèmes classiques de synchronisation (suite)



Problème des philosophes

```
void * philosophe(void * num) {
    int i = *(int *) num, nb = 2;
    while (nb) {
        /* penser */
        sem_wait(&mutex);
        phiState[i] = THINKING;
        sem_post(&mutex);
        sleep(1);

        /* essayer de manger */
        sem_wait(&mutex);
        phiState[i] = HUNGRY;
        test(i);
        sem_post(&mutex);

        /* attendre son tour */
        sem_wait(&semPhil[i]);

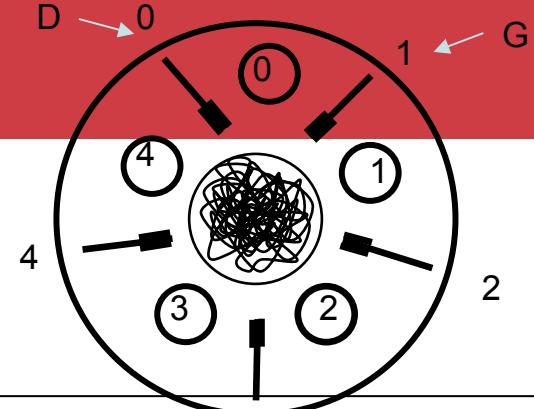
        /* à nous de manger */
        printf("philosophe[%d] mange\n", i);
        sleep(1);
        printf("philosophe[%d] a fini\n", i);
    }
}
```

// vérifie si le philosophe i peut manger

```
void test(int i) {
    if ((phiState[i] == HUNGRY) &&
        (phiState[G(i)] != EATING) &&
        (phiState[D(i)] != EATING)) {
        phiState[i] = EATING;
        sem_post(&semPhil[i]);
    }
}
```

/* laisser manger ses voisins */

```
sem_wait(&mutex);
phiState[i] = THINKING;
test(G(i));
test(D(i));
sem_post(&mutex);
nb--;
}
```



Problème des Lecteurs / Rédacteurs

Solution :

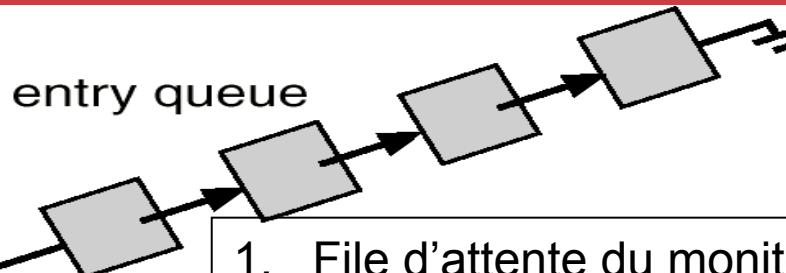
Semaphore tour=1, mutex=1, redact =1;

```
Redacteur () {  
    while(1) {  
        P(tour); // attendre son tour  
        P(redact); // assurer l'accès  
                    // exclusif à la base  
        V(tour);  
        Ecriture();  
        V(redact);  
    }  
}
```

```
Lecteur () {  
    while(1) {  
        P(tour); // attendre son tour  
        P(mutex) ;  
        Nbl++;  
        if (Nbl==1) P(redact);  
        V(mutex);  
        V(tour);  
        Lecture();  
        P(mutex);  
        Nbl--;  
        if(Nbl==0) V(redact);  
        V(mutex);  
    }  
}
```

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).



Exécution en
exclusion mutuelle
des méthodes

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 moniteur.

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)

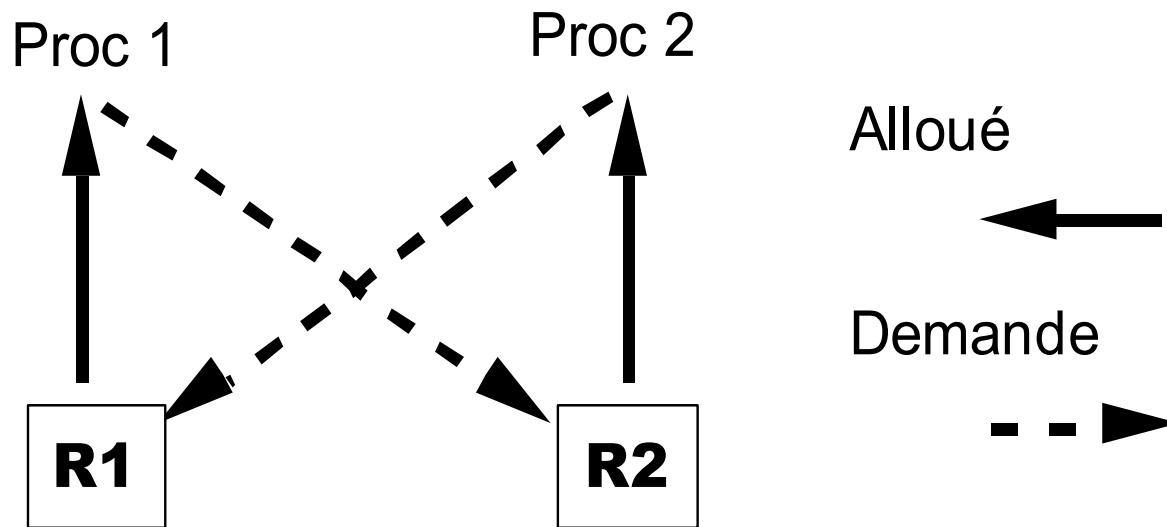
```
{  boolc nplein, nvide ; //variable de condition pour non plein et non vide  
  int compteur =0, ic=0, ip=0 ;  
  void mettre (int objet)    // section critique pour le dépôt  
  {  
      if (compteur==N) wait(nplein) ;  
      tampon[ip] = objet ;  
      ip = (ip+1)%N ;  
      compteur++ ;  
      if (compteur==1) signal(nvide) ;  
  }  
  void retirer (int& objet)  //section critique pour le retrait  
  {  
      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 ?



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.



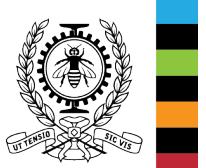
Solutions au problème d'interblocage

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

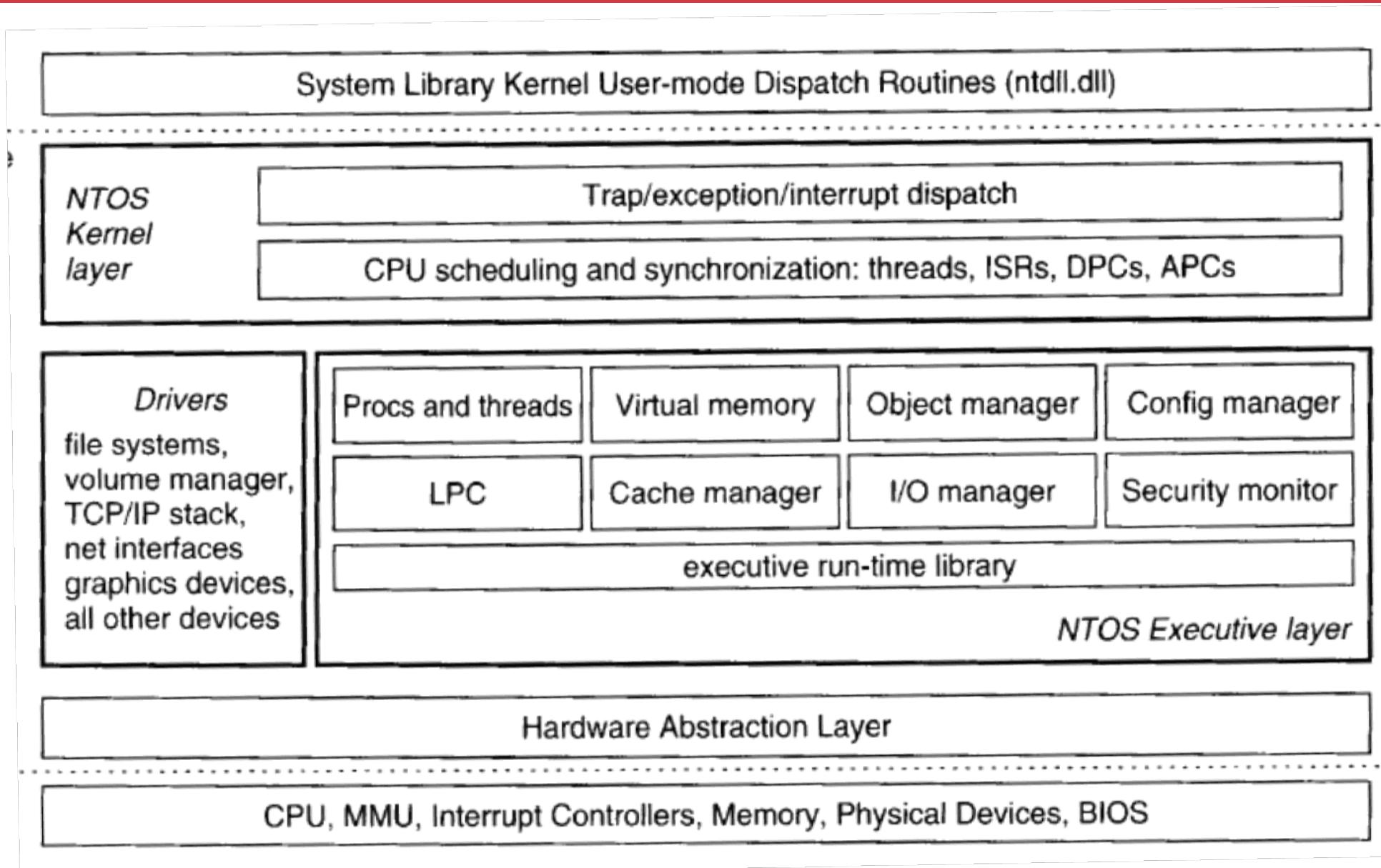
La prévention des interblocages

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.



Windows



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



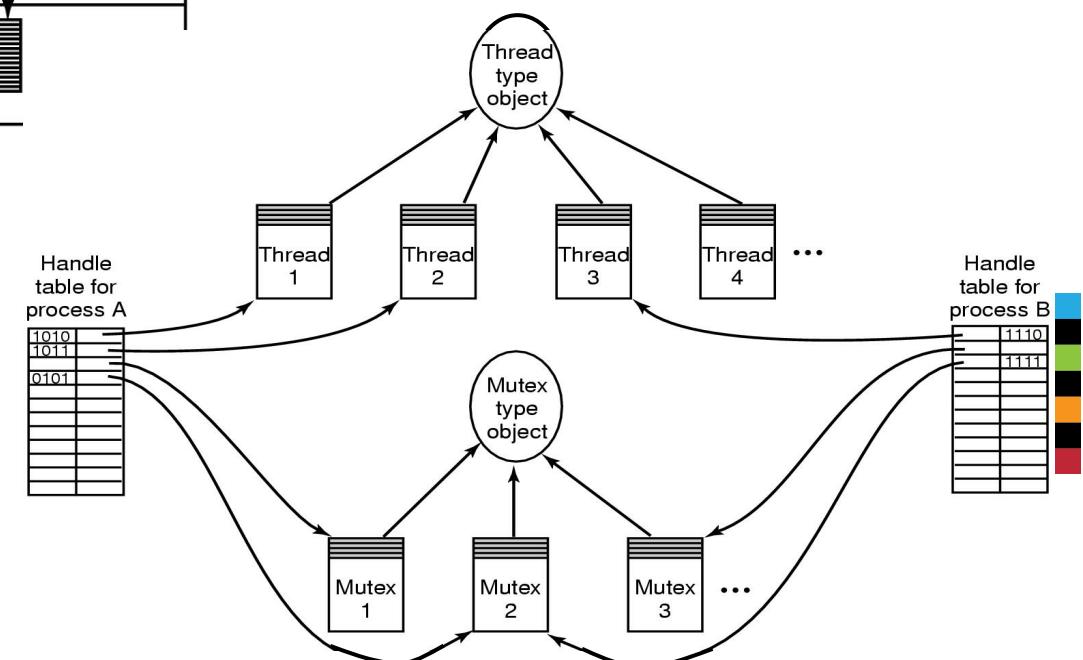
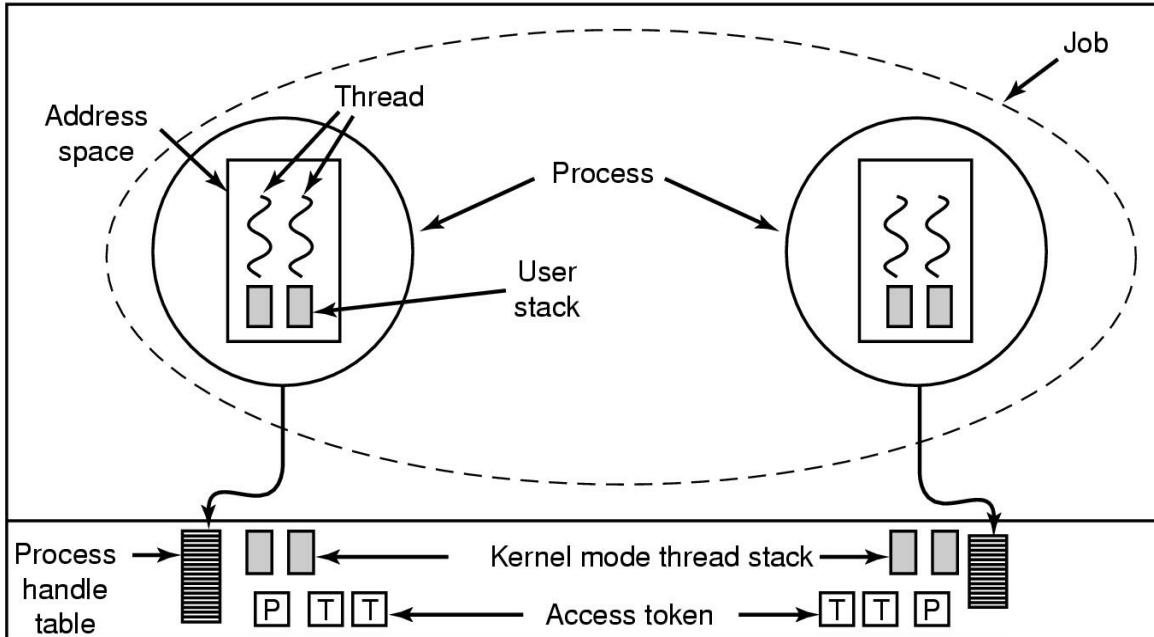
Windows

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



Windows Processus et threads



Windows

Processus et threads

```
BOOL WINAPI CreateProcess(  
    __in_opt    LPCTSTR lpApplicationName, // myProg.exe  
    __inout_opt  LPTSTR lpCommandLine,     // -> main (argc,argv[])  
    __in_opt    LPSECURITY_ATTRIBUTES lpProcessAttributes,  
    __in_opt    LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    __in        BOOL bInheritHandles, // permettre au fils d'accéder aux objets du père  
    __in        DWORD dwCreationFlags, // Ex: CREATE_NEW_CONSOLE  
    __in_opt    LPVOID lpEnvironment, // variables d'environnement  
    __in_opt    LPCTSTR lpCurrentDirectory, // répertoire courant  
    __in        LPSTARTUPINFO lpStartupInfo,  
    __out       LPPROCESS_INFORMATION lpProcessInformation  
);
```



Windows

Processus et threads

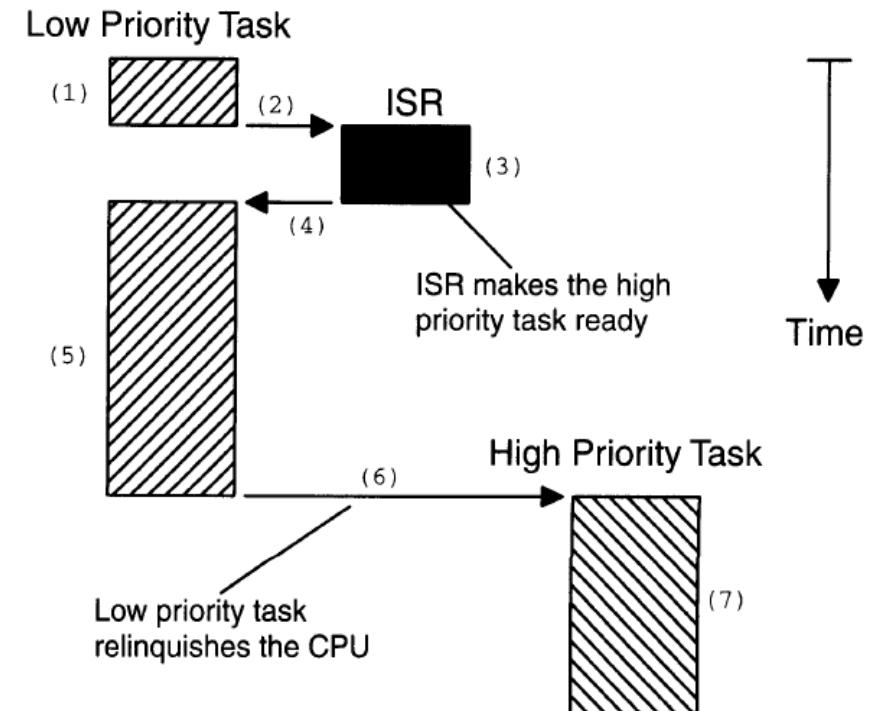
```
typedef struct _STARTUPINFO {
    DWORD cb;
    LPTSTR lpReserved;
    LPTSTR lpDesktop;
    LPTSTR lpTitle;
    DWORD dwX;
    DWORD dwY;
    DWORD dwXSize;
    DWORD dwYSize;
    DWORD dwXCountChars;
    DWORD dwYCountChars;
    DWORD dwFillAttribute;
    DWORD dwFlags;
    WORD wShowWindow;
 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

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



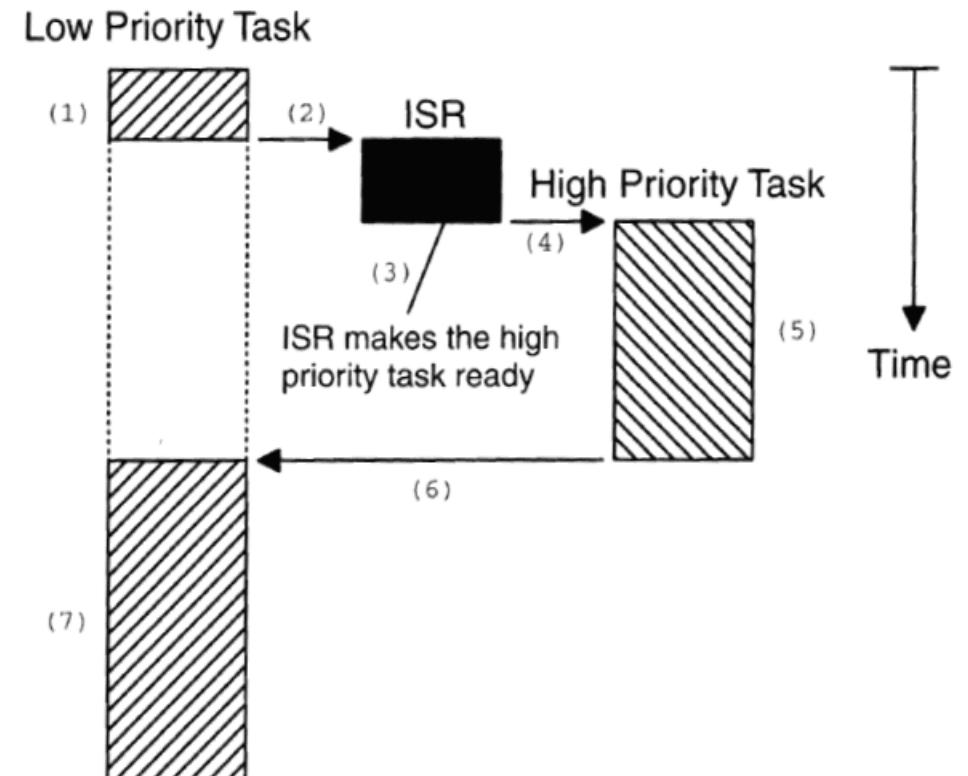
Ordonnancement de processus non préemptif

- 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 T₁,...,T_n 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_i = P_i 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_i = C_i / P_i Taux d'occupation processeur de la tâche T_i.

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 (exemple)

T1 : C1=12 P1=50

T3 : C3=10 P3 = 30

T2 : C2=10 P2 = 40

Sont-elles ordonnable RMA ?

0.24+0.25+0.33 > 78% On ne peut pas conclure

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

→ Diagramme de Gantt.



Exercice 1 : 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).

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.



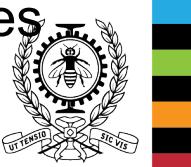
Exercice 1 : Gestion de la mémoire (Hiv16)

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 maintenant 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.

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 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 : 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 4 : Ordonnancement (Aut15)

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

Processus	Date d'arrivée (O _i)	Durée d'exécution (C _i)
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 5 : Ordonnancement (Hiv13)

Considérez un système d'exploitation monoprocesseur doté d'un ordonnanceur préemptif, à priorités. Ce système gère 16 niveaux de priorités, de 0 à 15, 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, qu'ils accèdent 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 6 : Gestion de la mémoire (Hiv14)

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 7 : Ordonnancement (Aut15 - suite)

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 toujours 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 aux sections critiques ainsi que les attentes (passives ou actives). Donnez le temps de séjour moyen pour chaque cas.

Exercice 8 : 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 8 : Windows (Hiv14)

```
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 9 : Interblocage (Aut15)

Un système gère 4 processus et 3 types de ressources. L'état courant du système est :

Processus	Alloc			Req			A (ressources disponibles)		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	1	0	2	0	1	0	1	1	x
P2	2	0	1	0	y	4			
P3	1	1	0	1	0	3			
P4	1	1	1	0	0	1			

Où Alloc indique les ressources allouées à chaque processus et Req donne pour chaque processus, les ressources nécessaires mais non encore obtenues.

- 1) Supposez que $x=2$ et $y =1$. À partir de cet état, donnez, s'il existe, un ordre des demandes d'allocation de ressources (non encore obtenues) qui mène vers un interblocage, dans le cas où l'évitement des interblocages n'est pas appliqué.
- 2) Supposez maintenant que l'algorithme du banquier est appliqué pour éviter les interblocages. Pour quelles plus petite valeur de x et plus grande valeur de y l'état courant est sûr
- 3) Supposez toujours que l'algorithme du banquier est appliqué pour éviter les interblocages, $x=2$, $y =1$ et que le processus P3 demande 2 ressources de type R3. Est-ce que le système va accepter cette demande ? Justifiez votre réponse.



Exercice 10 : Gestion de la mémoire

Caractéristiques du système :

Mémoire physique de 64 mots de 8 bits (64 octets)

Taille d'une page = 4 mots

Table des 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 des pages sur 8 bits:

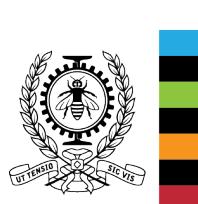
- 2 bits de contrôle : bit de présence, bit de référence (10,11)
si bit de présence =0 et bit de référence =1, la page est dans la zone de « swap »
- 6 bits pour l'adresse

Deux processus P1 et P2 sont en mémoire.

La table de premier niveau de P1 commence à 32

La table de premier niveau de P2 commence à 52.

Supposez que l'état courant de la mémoire physique est :



Exercice 10 : Gestion de la mémoire

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	35
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 10 : 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 10 : 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 10 : Gestion de la mémoire

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

20

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

Table de pages de P2

52

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

24

0	0	0
0	0	0
0	1	0
1	0	16

48

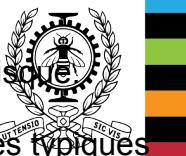
0	0	8
1	0	0
0	0	0
0	0	0



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.** Infonuagique. 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.



Annexe

<http://www5.in.tum.de/~huckle/bugse.html>

http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Authoritative_Account.html

Outils de vérification de codes C, C++, Java :

<http://www.valgrind.org/>

<http://lrbmc.org/index.html>

....

