CONSERVATOIRE NATIONAL DES ARTS et METIERS
Centre de Paris

Examen Corrigé NFP137 : Systèmes et applications concurrentes

3 juillet 2007

Exercice 1 : Gestion des ressources : L'algorithme du banquier.

Lorsqu'une tâche entre dans le système, elle doit déclarer le nombre maximum d'instances de chaque type de ressources qu'elle doit utiliser. Lorsqu'une requête utilisateur demande un ensemble de ressources, le système doit assurer si cette allocation laisse le système dans un état **sain.**
Les structures de données suivantes permettent de gérer l'algorithme du banquier :

- ✓ Disponible[m] contient le nombre de ressources disponibles de chaque type ; Disponible[j] = k signifie que k ressources de type j sont disponible.
- ✓ Max[n][m] indique le nombre maximum de demandes de chaque tâche. Max[i][j] = k signifie que la tâche Ti peut requérir au plus k instances de ressources de type Rj.
- ✓ Allocation[n][m] contient le nombre de ressources de chaque type actuellement alloué à chaque tâche. Allocation[i][j] = k indique que la tâche Ti a actuellement alloué k instances de ressources de type Ri.
- ✓ Besoin[n][m] indique le besoin restant en ressource pour chaque tâche. Besoin[i][j] = k signifie que la tâche Ti a besoin de k instances de plus des ressources Rj pour terminer sa tâche.

Considérez l'image suivante à un instant donné t0 du système :

t0	*Allocation*	*Max*	*Disponible*
	A B C D	A B C D	A B C D
T0	0 0 1 2	0 0 1 2	1 5 2 0
T1	1 0 0 0	1 7 5 0	
T2	1 3 5 4	2 3 5 6	
T3	0 6 3 2	0 6 5 2	
T4	0 0 1 4	0 6 5 6	

1) En utilisant l'algorithme du banquier donner le contenu de la matrice Besoin[5][4]
2) Le système est-il dans un état sain ?
3) Si la tâche T1 émet la requête (0,4,2,0), peut-elle être honorée immédiatement ?

1)

t0	Besoin
	A B C D
T0	*0 0 0 0*
T1	*0 7 5 0*
T2	*1 0 0 2*
T3	*0 0 2 0*
T4	0 0 4 2

2) On applique l'algorithme du banquier :
Work = 1 5 2 0
On cherche i tel que Besoin i <= Work
i=0
On calcule le nouveau Work
Work = 1 5 3 2
On cherche i tel que Besoin i <= Work
i=3
Work = 1 11 6 4
On cherche i tel que Besoin i <= Work
i=1
Work = 2 11 6 4
On cherche i tel que Besoin i < =Work
i=2
Work = 3 14 11 8
On cherche i tel que Besoin i <= Work
i=4
Work = 3 14 12 12
La séquence T0, T3, T1, T2, T4 satisfait le critère d'état sain.

3) T1 = (0,4,2,0) alors la table Allocation devient :

t1	Allocation
	A B C D
T0	*0 0 1 2*
T1	***1 4 2 0***
T2	*1 3 5 4*
T3	*0 6 3 2*
T4	0 0 1 4

Disponible = 0 1 0 0

t1	Besoin
	A B C D
T0	*0 0 0 0*
T1	***1 4 2 0***
T2	*1 0 0 2*
T3	*0 0 2 0*
T4	0 0 4 2

Work = 0, 1, 0, 0 : Il n'existe pas de ligne de Besoin tel que Besoin i <= Work
Donc même si les ressources existent, cette requête conduit à un état non sain.

Exercice 2 : Synchronisation

On considère l'algorithme principal de la version suivante du paradigme de synchronisation « lecteurs – rédacteurs »

La structure d'un flot lecteur :

```
while (true) {
        P (read) ;
                readcount ++ ;
                if (readcount == 1)  P (wrt) ;
           V (read) ;

              // la lecture est effectuée

         P (read) ;
                readcount  - - ;
                if (readcount  == 0)  V (wrt) ;
         V (read) ;
            utiliser_donnees();
}
```

La structure d'un flot rédacteur :

```
while (true) {
            creer_donnees() ;
         P (wrt) ;

             //  l'écriture est effectuée

         V (wrt) ;
}
```

1) Donner les types et les initialisations des variables *read, wrt et readcount.*
2) Expliquer le type de priorité entre les lecteurs et les rédacteurs induit par un tel algorithme.
3) Transformer cet algorithme pour qu'il y ait équité entre les lecteurs et les rédacteurs.

1) Mutex read ;
Mutex Wrt ; int readcount = 0 ;
E0(read,1) ; E0(wrt,1) ;

2) Priorité aux lecteurs : les rédacteurs n'ont pas la possibilité d'interrompre une rafale de requête provenant des lecteurs.

3)

Toutes les requêtes d'accès sont séquentialisées par le sémaphore redact .

La structure d'un flot lecteur :

```
Mutex redact ;
E0(redact, 1) ;

while (true) {
        P(redact) ;
      P (read) ;
            readcount ++ ;
            if (readcount == 1)  P (wrt) ;
        V (read) ;
        V(redact) ;

          // la lecture est effectuée

      P (read) ;
            readcount  - - ;
            if (readcount  == 0)  V (wrt) ;
      V (read) ;
        utiliser_donnees();
}
```

La structure d'un flot rédacteur :

```
while (true) {
        creer_donnees() ;
        P(redact) ;
      P (wrt) ;
        V(redact) ;

        //  l'écriture est effectuée

      V (wrt) ;
}
```

Problème : Gestion mémoire

Dans les systèmes informatiques modernes l'unité centrale est partagée par un ensemble de tâches ; pour assurer une bonne performance à chacune, le système doit maintenir plusieurs tâches en mémoire ; ce partage de mémoire induit une gestion qui nécessite des ressources matérielles et des algorithmes systèmes.

Les algorithmes de gestion mémoire pour un système multiprogrammé sont très divers, du simple système mono utilisateur au système segmenté paginé. Chaque adresse produite par le processeur doit être validée et transformée en une adresse physique ; cette transformation est réalisée par un dispositif matériel, l'unité de gestion mémoire (MMU).

La méthode d'allocation contigüe nécessite de diviser la mémoire en partitions de taille fixe : le système gère alors une table indiquant les espaces mémoires libres et les espaces occupés. A l'initialisation, l'ensemble de la mémoire est disponible ; lorsqu'une tâche nécessite de la mémoire, le système cherche un espace suffisant pour allouer la mémoire demandée ; cet espace est déduit de l'espace libre et toute la mémoire nécessaire est allouée à la tâche. Quand la tâche termine elle libère l'espace mémoire correspondant.

La pagination est un schéma de gestion mémoire qui permet d'allouer des espaces non contigües et résout en partie les problèmes posés par la gestion précédente.

Les développeurs perçoivent un programme en cours d'exécution comme un ensemble structuré d'éléments et non comme une liste d'adresses mémoire identiques : l'espace programme, l'espace données statiques, l'espace pile, l'espace bibliothèque, l'espace données dynamique ; chacun de ces espaces est de taille différente. **La segmentation** est un schéma de gestion mémoire qui supporte cette vision utilisateur.

Le schéma segmenté-paginé combine les deux approches précédentes pour rendre plus efficace la gestion mémoire d'un grand nombre de tâches partageant une mémoire centrale importante.

En comparant les stratégies de gestion mémoire les plus courantes (allocation contigüe, pagination, segmentation, segmentation paginée) les critères suivants sont impactés :

- le matériel nécessaire
- l'allocation des informations (données, code, pile…) en mémoire physique
- la performance
- la fragmentation
- l'extension d'espace (swapping)
- le partage de données et de code
- la protection

1) Pour chacun des critères précédents, expliquer clairement les conséquences des différentes stratégies de gestion mémoires évoquées ci-dessus.

	Contigüe	Paginée	Segmentée	Paginée-segmentée
Matériel	Un ensemble de registres (base-limites) par partition	Une table des pages dans la MMU	Une table des segments	Une ou plusieurs tables des pages, une table des segments
Allocation mémoire	La mémoire doit être périodiquement recompactée pour récupérer les « trous »	De la taille d'une page : allocation est dynamique par pages	De la taille de chaque segment dont les longueurs sont définies statiquement	Allocation dynamique de pages dans des segments fixes.
Performances	Une addition pour former l'adresse : très rapide	Conversion virtuelle réelle rapide si les tables sont dans des registres ou mémoires locales. Peu performant si les tables sont sur disque	Une anticipation par une TLB est nécessaire pour ne pas dégrader les performances	La segmentation rajoute une indirection de plus par rapport au cas précédent.
Fragmentation	Rapidement, de nombreux espaces non utilisables se créent au fur et à mesure des allocation et désallocation : la mémoire devient très fragmentée	Fragmentation interne à un espace maximum de la taille d'une page	Fragmentation externe de la taille de l'espace réservé pour un segment	Supprime la fragmentation précédente en limitant l'espace qui peut être perdu à la taille d'une page
Extension d'espace	Swapping de toute la mémoire associée à la tâche	Echange avec le disque d'une seule page, à la demande.	Gestion d'espace pour le segment qui le nécessite	Idem pagination.
Partage de données et de code	Impossible	Possible en indiquant le mode dans la table de pages	Possible, par segments	Possible, soit par segment, soit par page
Protection	Impossible	Des sections de programmes peuvent être déclarées en lecture seule, en lecture/écriture ou en exécution	Les sections précédentes sont des segments	Idem cas précédent.

On considère la gestion mémoire du processeur Pentium résumée ainsi :

L'architecture de la gestion mémoire de l'Intel Pentium supporte à la fois la segmentation et la segmentation-paginée. L'UC fournit une **adresse logique**, traité par le système de segmentation qui produit une **adresse linéaire** pour chaque adresse logique ; cette adresse linéaire est transférée au mécanisme de pagination qui engendre une **adresse physique**.

Cette architecture autorise des segments dont la taille peut atteindre 4 GOctets et le nombre maximum de segments par tâche est de 16 K.

L'espace logique d'une tâche est divisé en deux partitions :

- La première partition peut atteindre 8 K segments privés pour la tâche
- La seconde partition, jusqu'à 8K segments, est partagée par toutes les tâches.

La LDT (Local Descriptor Table) mémorise les informations de la première partition, la GDT (Global Descriptor Table) mémorise les informations de la seconde.

Chaque entrée de ces tables est un descripteur de segments sur 8 octets, mémorisant l'adresse de base du segment, sa taille et des informations de contrôle.

L'adresse logique est composée de deux champs : le sélecteur sur 16 bits et un déplacement sur 32 bits.
Le sélecteur contient 3 champs : le N° de segment (13 bits), l'indication LDT ou GDT (1 bit), et une information de protection (2 bits).

Le processeur possède 6 registres segments, permettant à une tâche de gérer directement 6 segments d'information ; il contient en outre 6 registres internes pour mémoriser le contenu de la LDT ou de la GTD : ainsi, la machine n'a pas à accéder à la mémoire pour lire un descripteur.

L'adresse linéaire produite est ainsi formée :

- Le registre segment désigne une entrée dans la LTD ou la GTD.
- Les informations de base et de limite sont utilisées pour engendrer l'adresse linéaire.
- La limite est utilisée pour un test de validité qui peut remonter une interruption vers le système si erreur.
- La valeur du déplacement est ajoutée à la base pour produire l'adresse sur 32 bits.

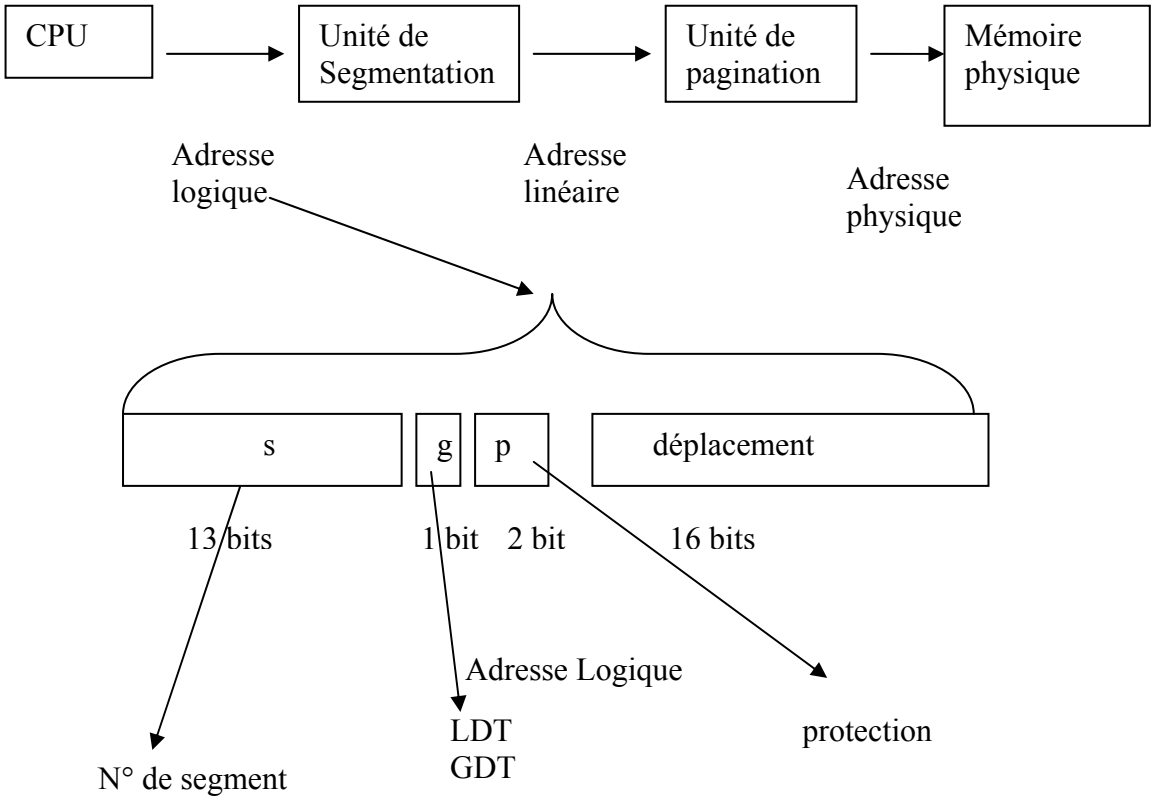
Cette adresse linéaire est envoyée au système de pagination pour produire l'adresse physique :
Le Pentium utilise des pages de 4KO ou 4MO. Un bit de type est présent dans l'entrée de la table des pages pour signifier cette alternative.
Pour des pages de 4KO, un mécanisme à 2 niveaux est réalisé :

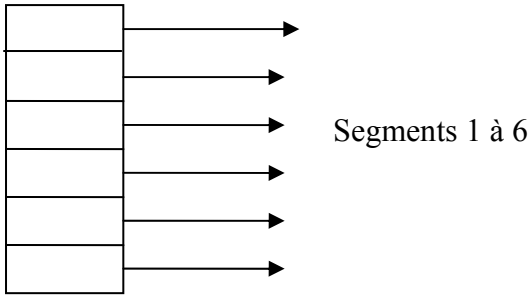
- Les 10 premiers bits désignent un répertoire de page ; ce répertoire est lui-même désigné par le contenu d'un registre nommé CR3, attaché à une tâche.
- Une entrée du répertoire précédent désigne une table de pages ; les 10 bits suivants de l'adresse linéaire sont rajoutés pour déterminer l'entrée à considérer dans la table des pages.
- Les 12 derniers bits désignent un déplacement pour identifier l'octet adressé dans la page.

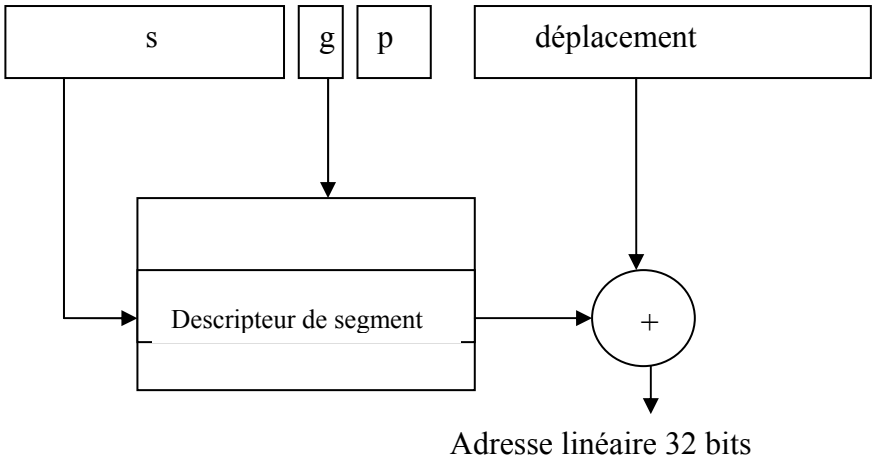
Pour des pages de 4MO, la table des pages n'existe pas ; 1024 pages de 4MO sont alors accessibles, le déplacement sur 22 bits permettant d'accéder à l'octet considéré.

Pour améliorer l'utilisation de la mémoire physique, les tables du système de pagination peuvent être swappées sur disque : un bit de validité est utilisé dans l'entrée de la table des pages pour savoir si la table est en mémoire ou sur disque ; dans ce dernier cas les 31 bits restants sont utilisés pour spécifier l'adresse disque, ce qui permet de charger ces tables seulement lorsque qu'elles sont nécessaires.

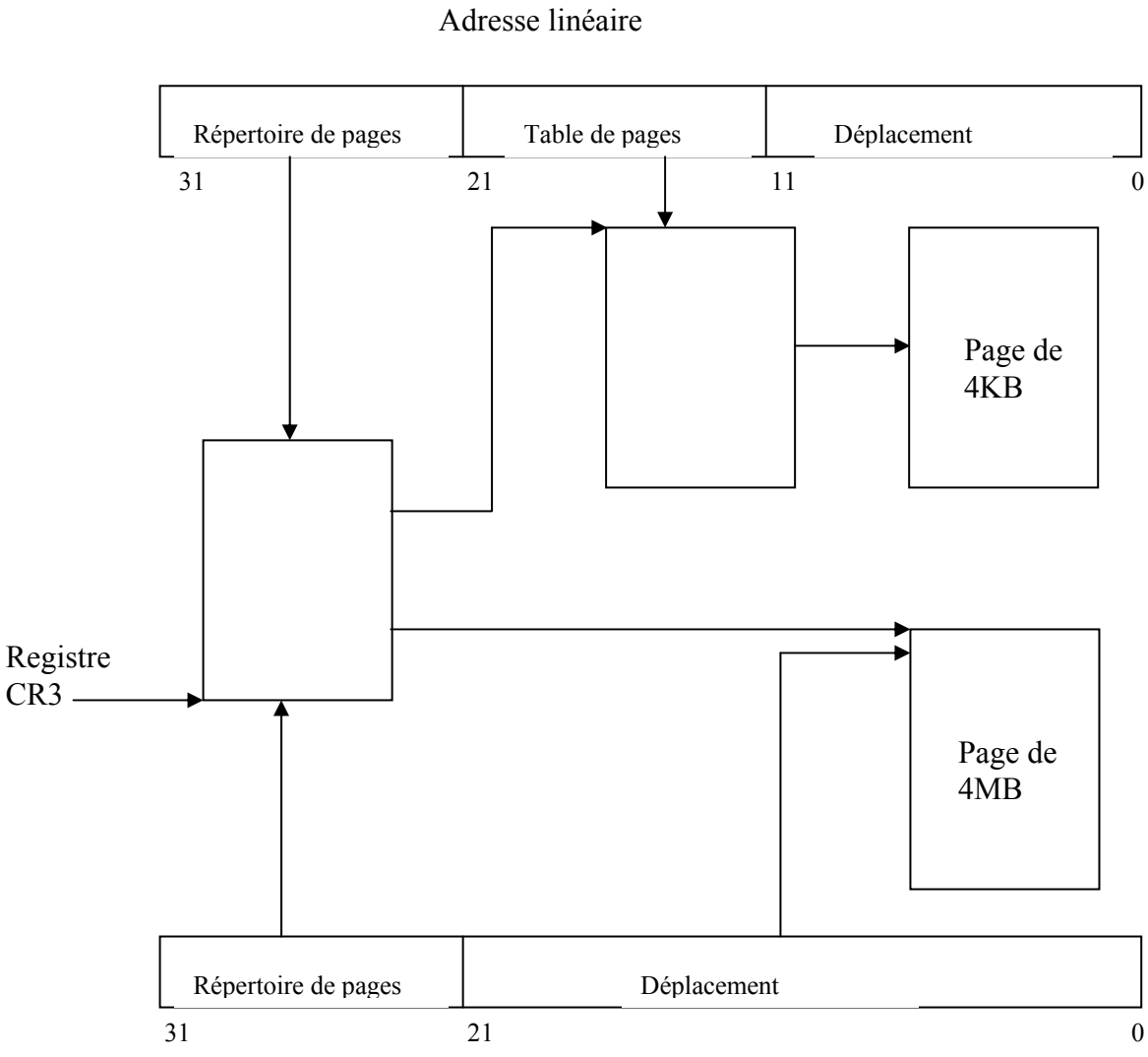
2) Décrivez toutes les actions entreprises par le processeur Pentium pour produire une adresse physique en **représentant graphiquement** les différentes structures mises en œuvre pour les traductions d'une adresse logique en une adresse réelle.


CPU
Unité de Segmentation
Unité de pagination
Mémoire physique
Adresse logique
Adresse linéaire
Adresse physique
s
g
p
déplacement
13 bits
1 bit
2 bit
16 bits
Adresse Logique
LDT
GDT
protection
N° de segment


Segments 1 à 6
6 Registres segments



Segmentation du Pentium



Pagination du Pentium

3) Quels sont les avantages pour le système d'exploitation d'un processeur offrant de telles ressources ?

L'allocation mémoire est gérée par le matériel, ce qui permet de n'avoir en émoire que les informations nécessaires à l'exécution de la tâche et à l'instant où elles sont nécessaires.

La mise en œuvre de la protection est réalisée au niveau du segment ; il n'y a pas de fragmentation de la mémoire supérieure à 4KB ; des applications nécessitant une grande quantité de mémoire, peuvent disposer d'un incrément mémoire de 4 M octets ; de ême des applications très interactives, ne nécessitant que peu de mémoire peuvent transférer des incréments de 4 K octets.

Les segments peuvent être partagés par toutes les tâches, autorisant des codes ré-entrants ou des partages de données, tous les descripteurs de ces segments étant accessibles par la GTD.

Les tables nécessaires à la conversion des adresses sont chargées dynamiquement, à la demande, ce qui permet de ne pas mobiliser de ressources qui ne seraient pas utilisées.

La performance de la traduction est assurée par des opérations simples se résumant à des accès mémoires locales (tables), registres et des additions ou des tests.