FACULTE DES SCIENCES ET DE LA TECHNOLOGIE
DEPARTEMENT MATHEMATIQUES & INFORMATIQUE L2 Informatique (SI) 2020/2021

Matière : systèmes d'exploitation 1 **(Operating System)**
Chapitre 3 : Gestion de la mémoire (coeff=3 ; crédit=5)

Plan de cours

1 Gestion de la mémoire principale
1.1 Gestionnaire de la mémoire
1.2 Chargement dynamique
1.3 La gestion de la mémoire dans les systèmes mono programmés.
1.4 La gestion de la mémoire dans les systèmes multi programmés
1.4.1 Gestion de la mémoire avec la technique des partitions fixes
1. La protection
2. L'ordonnancement des travaux
3. Choix de la taille des partitions
4. Fragmentation de la mémoire
1.4.2 Gestion de la mémoire avec la technique des partitions variables
1.4.2.1 Allocation d'une partition
1.4.2.2 Libération d'une partition
1.4.2.3 Stratégies de placement (ou d'allocation)
1.4.2.3.1 First – fit
1.4.2.3.2 Best – fit
1.4.2.3.3 Worst – fit
1.4.3 Fragmentation et compactage

2 Gestion de la mémoire virtuelle
2.1 Adresses logiques et adresses physiques
2.2 Espace d'adressage logique et espace d'adressage physique
2.3 Le concept de mémoire virtuelle
2.4 La pagination
2.4.1 Traduction des adresses virtuelles en adresses réelles
2.4.2 Taille de la table de pages
2.4.3 Le choix de taille de page
2.5 La segmentation
2.6 Segmentation avec pagination

3 La pagination à la demande
3.1 Détection de défaut de page
3.2 Traitement de défaut de page
3.3 Remplacement de page
3.3.1 Evaluation des algorithmes de remplacement
3.3.2 Les algorithmes de remplacement
3.3.2.1 L'Algorithme ***FIFO***
3.3.2.2 L'Algorithme optimal (***OPT ou MIN***)
3.3.2.3 L'Algorithme ***LRU*** (Least Recently Used)
3.3.2.4 L'Algorithme ***NRU*** (Not Recently Used; non récemment utilisé)

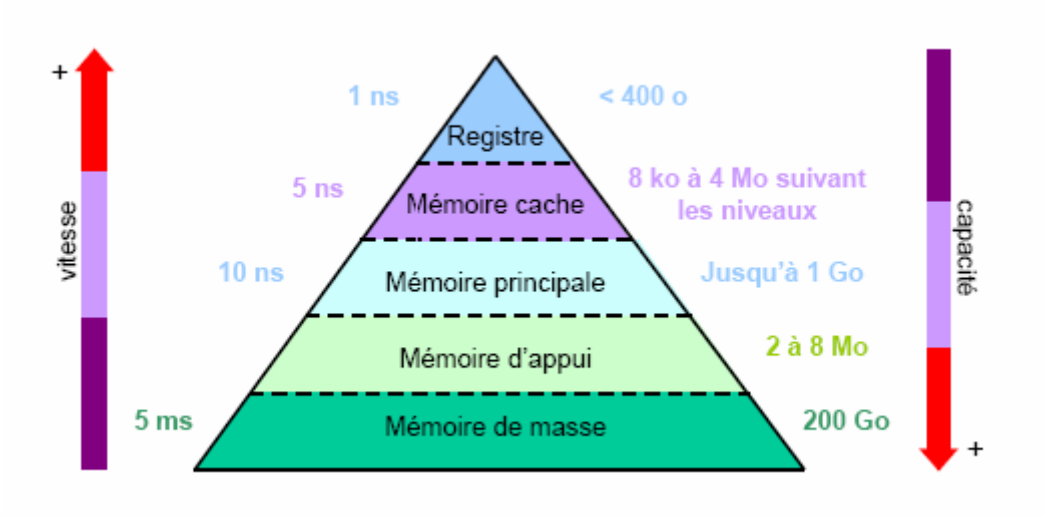
Introduction

près l'UC, la mémoire centrale est le deuxième composant principal de l'architecture de Von Neumann. Son rôle est de conserver les informations (données et programmes) manipulées par l'UC et de les restituer en cas de besoin. Ainsi, pour qu'un programme puisse être exécuté sur des données par l'UC, les (programme et données) doivent être chargés en MC d'abords. Le chargeur cherche un espace libre suffisant pour contenir P.

Pour maximiser le rendement de l'UC dans un système multiprogrammé, plusieurs questions se pose concernant :

- Comment partager les données entre les processus ?
- Comment protéger l'espace mémoire de chaque processus?
- Qui et comment faire l'allocation d'espace mémoire aux process?
- Comment et quand la Restitution de l'espace libéré par le process se fait?

Pour répondre à ces exigences, le SE doit se doter d'un module spécial de gestion de la MC appelé Gestionnaire de la MC -Memory Management Unit MMU.



Objectifs de la gestion de la mémoire centrale

1. Organisation de la MC : structuration des données en un ensemble de mots mémoires et effectuer une traduction des adresses logiques des données à des adresses physiques.

2. Allocation et Réallocation : attribution d'un espace mémoire, éventuellement *libéré, à un processus demandeur*

3. Libération de l'espace et restitution : Forcée ou volontaire.

4. Partager des informations (données et programmes) entre les processus utilisateurs

5. Protection de l'espace mémoire des processus et du SE.

6. Surpasser les limites physiques de la capacité de la MC.

Ces objectifs doivent être accomplis d'une manière efficace, par augmentation des performances du système informatique et la transparence pour l'utilisateur.

1 Gestion de la mémoire principale

Tout processus existant dans le système a besoin d'espace mémoire pour y charger son programme et ses données. Car le programme d'un processus ne peut être exécuté que s'il est chargé en mémoire principale. Le processeur prélève les instructions à exécuter à partir de la mémoire centrale. La procédure normale consiste à charger le programme en mémoire centrale, à l'aide d'un programme système appelé **chargeur**.

Le programme chargé peut être :

- ***En absolu*** dans ce cas le chargement s'effectue à des emplacements fixes de la mémoire,

- ***Translatable (ou relogeable***) dans ce cas, le programme doit subir une opération de translation des adresses.

Le programme peut être chargé entièrement ou partiellement selon la technique de gestion mémoire utilisée.

Il existe plusieurs techniques de gestion de la mémoire. Selon l'architecture de la machine concernée, certaines techniques ne nécessitent aucun dispositif matériel particulier ; d'autres, au contraire, exigent une architecture plus élaborée. Le choix d'une technique dépend de plusieurs facteurs et essentiellement de la conception de la machine.

1.1 Gestionnaire de la mémoire

C'est un ***module système*** dont la fonction est de gérer la mémoire principale. Il doit garder en permanence des informations concernant l'espace libre et occupé de la mémoire. Le gestionnaire alloue de la mémoire aux processus qui en ont besoin et récupère la mémoire libérée par un processus.

La mémoire principale étant limitée, le gestionnaire ne peut pas charger tous les processus en mémoire centrale. Pour satisfaire les besoins des processus, le gestionnaire doit décharger certains programmes de la mémoire principale et charger d'autres programmes à partir du disque. Il est à noter que les programmes déchargés de la mémoire principale sont sauvegardés sur disque (mémoire secondaire ou auxiliaire).

Les principales fonctions d'un gestionnaire de la mémoire sont les suivantes :

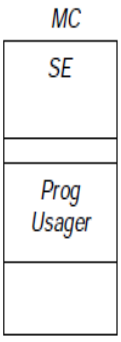
- Allouer / libérer l'espace de la mémoire réelle ;
- Charger / décharger (sauvegarder) les programmes des processus ;
- Protéger les programmes et les données des processus ;
- Partager des programmes et / ou des données entre plusieurs processus.

La correspondance entre adresses ***virtuelles (logiques)*** et ***adresse réelles (physique)*** est effectuée par un dispositif matériel que l'on appelle unité de gestion mémoire (***MMU : Management Memory Unit***). Cette correspondance s'appelle aussi ***mapping d'adresses***.

1.2 Chargement dynamique

Pour optimiser l'utilisation de la mémoire, on peut utiliser le chargement dynamique ; une routine ou une procédure n'est chargé qu'au moment du premier appel. Avec cette méthode, une procédure non utilisé n'est jamais chargée en mémoire centrale : donc gain d'espace mémoire.

1.3 La gestion de la mémoire dans les systèmes mono programmés



Dans ces systèmes la mémoire est divisée en ***deux parties*** : Une partie réservée au système, l'autre partie sera allouée à l'utilisateur (ce système est caractérisé par la présence d'un ***seul processus utilisateur*** prêt dans la MC). Cette approche a été utilisée dans l'un des premiers systèmes.

Cette technique en voie de disparition est limitée à quelques micro-ordinateurs.

Elle n'autorise qu'un seul processus actif en mémoire à un instant donné.

1.4 La gestion de la mémoire dans les systèmes multi programmés

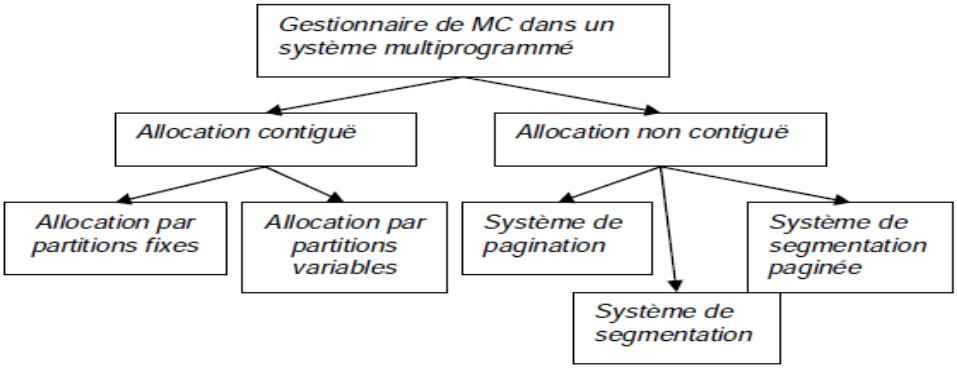


Figure Les approches de gestion de mémoire centrale dans un système multiprogrammé

1.4.1 Gestion de la mémoire avec la technique des partitions fixes

Dans un système à partitions de taille fixe, la mémoire est partagée de manière statique en un nombre fixe de ***zones ou partitions***. Les tailles et les adresses début de ces partitions sont définies lors de la génération de système ou bien par l'opérateur, au moment du chargement du système.

Programme 1	Partition 1
Programme 2	Partition 2
Programme 3	Partition 3
Programme 4	Partition 4

Cette technique permet la coexistence (simultanéité) de plusieurs programmes en MC, ce qui permet une meilleure utilisation des ressources mémoire principale et processeur central.

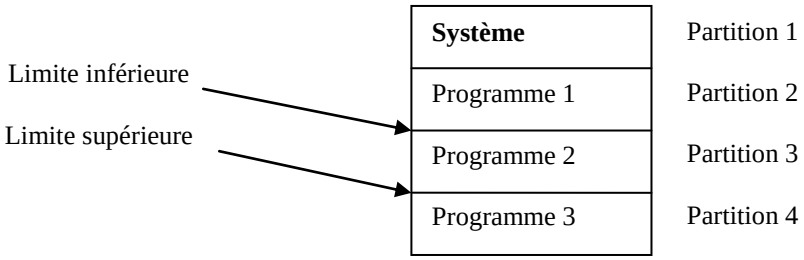
Pendant son exécution un programme ne doit pas accéder en dehors de la partition qui lui est allouée *sinon, un déroutement est généré : "* ***accès à une zone mémoire interdite*** *".*

1. **La protection**

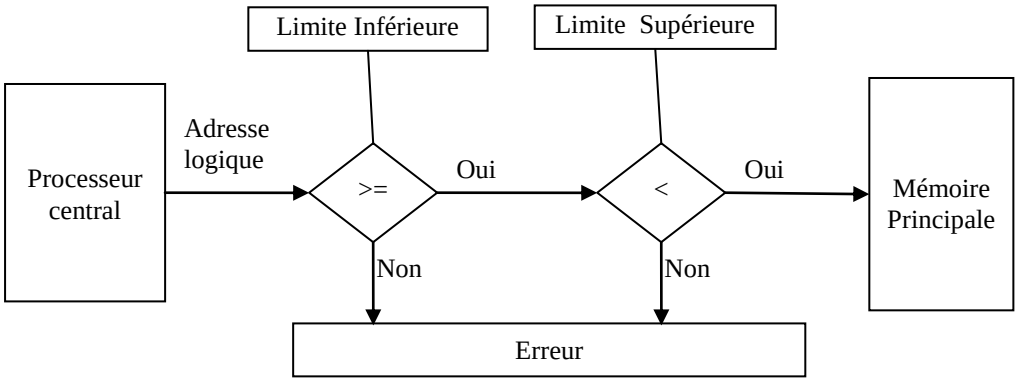
La coexistence du système et de plusieurs programmes en mémoire principale soulève le problème de protection du système et des programmes utilisateurs. La protection des programmes est réalisée au moyen de deux registres. Ces deux registres définissent la **limite inférieure** et la **limite supérieure** des adresses qui peuvent être utilisées par un programme. Ces limites peuvent être définies de deux manières :

Registre limite :

La plus petite et la plus grande adresse de la partition sont chargées dans des ***registres limite***.



Toutes les adresses de l'utilisateur sont comparées aux registres limite.



*Erreur : générer un **déroutement** (Accès à une zone mémoire protégée)*

La translation des adresses est faite de manière statique à l'assemblage / compilation ou au chargement du programme. ***Adresse physique = adresse logique + registre de base***

Swapping (ou méthode de va et vient)

Dans le mono programmé, le processus qui occupe un espace mémoire, il le gardera jusqu'à sa terminaison. Pour palier à ce problème, une libération temporaire de l'espace occupé pour donner l'occasion aux processus en attente dans le disque d'occuper la mémoire centrale.

Le principe de Swapping se base sur l'état d'un processus, si le processus P1 se bloque suite à une E/S, un autre processus P2 est chargé pour s'exécuter.

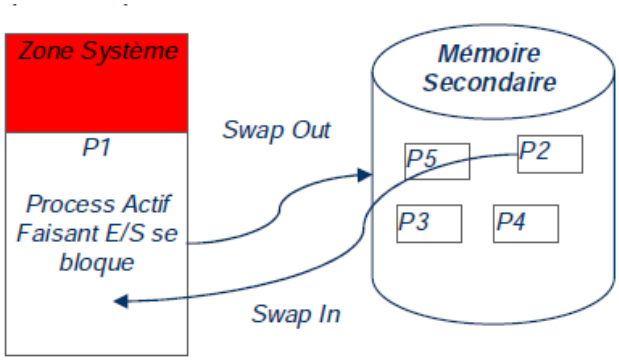
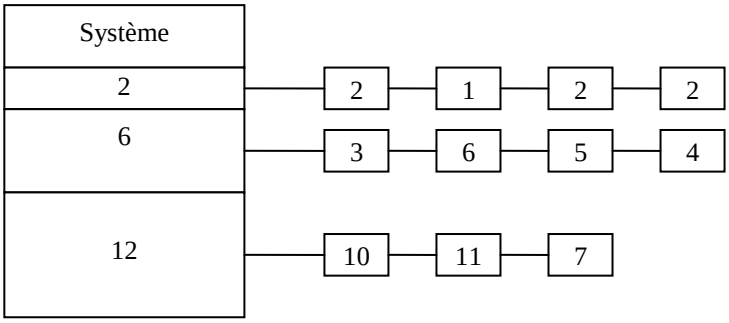


Figure Technique de swapping

2. L'ordonnancement des travaux

Tout travail (job) doit faire une demande explicite de l'espace mémoire nécessaire à l'exécution des programmes qui le composent ; cette demande est exprimée au moyen d'un langage de commandes. Le gestionnaire de la mémoire peut utiliser une file de travaux (jobs) par partition.



- Une autre approche consiste à mettre tous les travaux dans la même file d'attente et affecter un travail à la première partition dont la taille est supérieure ou égale à la taille demandée.

- Une autre variante consiste à utiliser la technique des partitions fixes avec la technique de *swapping*, qui consiste à stocker temporairement sur disque l'image d'un processus, afin de libérer de la place en MC pour d'autres processus.

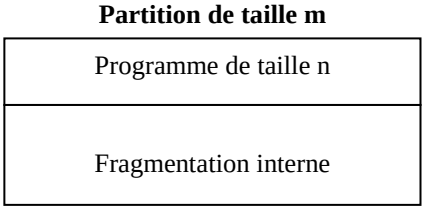
3. Choix de la taille des partitions

Les tailles des partitions sont définies en fonction de la nature des travaux à exécuter et des statistiques d'utilisation de la machine.

4. Fragmentation de la mémoire

On distingue 2 types de fragmentations :

Fragmentation interne : le choix des tailles des partitions doit être fait de telle sorte qu'il minimise la fragmentation interne de la mémoire principale. On dit qu'il y a fragmentation interne quand un programme de taille **n** est chargé dans une partition de taille **m** avec **m > n** : fragmentation interne = taille de la partition – taille du programme ou **(m – n).**

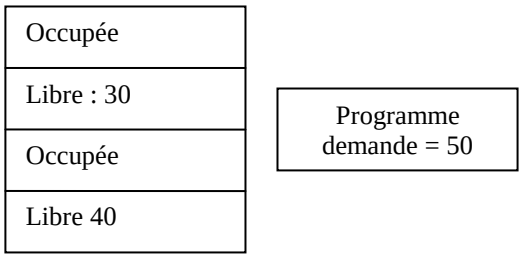


Fragmentation externe :

On dit qu'il y a **fragmentation externe**, si un programme de taille **n** est en attente de la mémoire alors qu'il existe un **espace mémoire total** suffisant pour satisfaire la requête (demande) du programme mais cet espace n'est pas contigu : il est constitué de plusieurs petites partitions (la somme des tailles des partitions libres >= taille du programme).

Soit un programme qui demande un espace mémoire 50 unités et une mémoire avec 2 partitions libres de taille respective 30 et 40 unités.

La requête de ce programme ne peut pas être satisfaite, car il n'existe pas de partition libre dont la taille est supérieure ou égale à la taille demandée par le programme.



1.4.2 Gestion de la mémoire avec la technique des partitions variables

Le problème avec la technique des partitions fixes est de déterminer les tailles des partitions qui minimisent la fragmentation interne de la mémoire principale. La solution à ce problème consiste à utiliser des partitions dont les tailles varient dynamiquement. Une partition est définie par sa taille et son adresse.

Au début, toute la mémoire est disponible (libre) pour les processus utilisateurs. Elle est constituée d'une seule partition ; les partitions sont créées au fur et à mesure de l'exécution des processus.

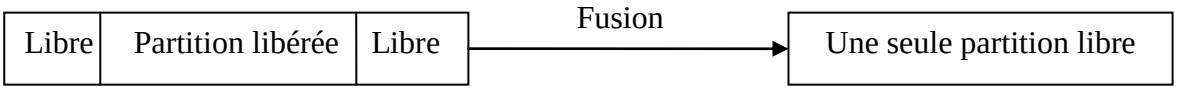
1.4.2.1 Allocation d'une partition

L'allocation d'une partition de taille **t** à un processus, consiste à trouver une zone de taille convenable parmi les zones (partitions) libres. La taille de la partition choisie doit être supérieure ou égale à la taille **t** demandée par le processus. Si la partition sélectionnée a une taille trop grande, elle est découpée en deux : une partie de taille **t** est

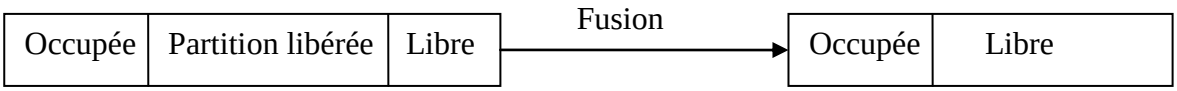
allouée au processus ; l'autre est insérée dans la liste des partitions libres (c'est ce que l'on appelle un éclatement de partition).

1.4.2.2 Libération d'une partition

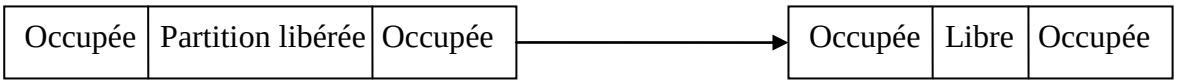
Quand un processus se termine, il libère sa partition qui est de nouveau placée dans la liste des zones libres. La libération d'une partition peut créer 3 situations différentes selon que la partition libérée est entourée de :
Deux partitions libres,



Une partition libre et d'une partition occupée,



Deux partitions occupées.



Chaque fois que cela est possible, il est utile de regrouper (fusionner) les partitions contiguës dès la libération. Cette fusion consiste à créer, à partir de plusieurs partitions contiguës, une seule partition de grande taille.

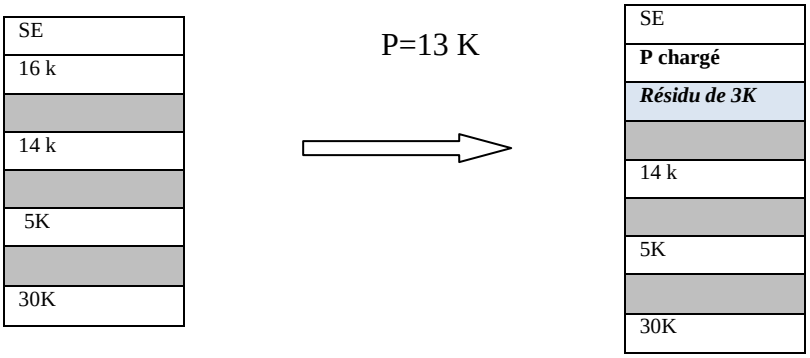
1.4.2.3 Stratégies de placement (ou d'allocation)

Le principal problème de cette technique d'allocation est le choix d'une partition libre de taille convenable. Il existe plusieurs stratégies de sélection d'une partition ; les plus importantes sont first – fit, best – fist et worst – fit.

1.4.2.3.1 Stratégie du 1er qui convient (First – fit)

Dans cette stratégie, on alloue ***la première partition*** dont la taille est ***suffisamment grande***. La recherche s'arrête dès que l'on trouve une zone libre de taille assez grande.
La recherche peut commencer au début de la liste ou là où la dernière recherche a fini (algorithme next – fit). La liste est ***ordonnée selon les adresses croissantes***.

Exemple :
Soit le schéma de la mémoire suivant avec 4 partitions libres. On désire allouer de l'espace un programme P de taille 13KO.



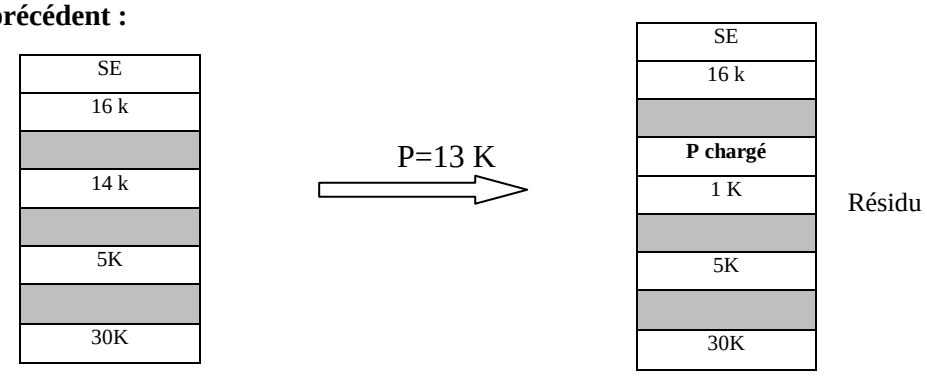
Avantage : stratégie rapide en temps d'exécution.

Inconvénients : on peut créer des partitions qui ne sont pas intéressantes (pas importantes : Exemple : 3k), qui ne seront pas être jamais utilisées. C'est la ***fragmentation interne***.

1.4.2.3.2 Stratégie du meilleur qui convient (Best – fit)

Allouer la partition dont la taille est ***la plus proche*** de la taille demandée. La recherche est effectuée dans toute la liste, à moins que la liste soit ordonnée par ordre ***croissant des tailles des partitions libres***.

Exemple précédent :



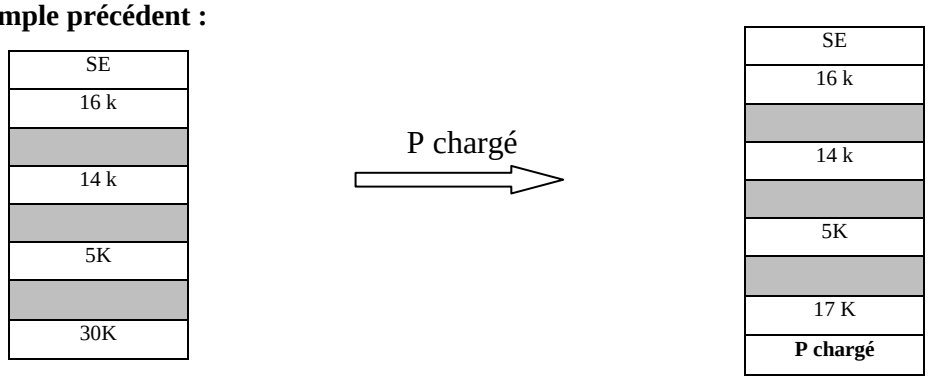
Avantage : stratégie qui tente de ***réduire*** la fragmentation de la mémoire (***résidus de petites tailles***)

Inconvénient : il faut connaitre les tailles de chaque partition libre, et maintenir une liste des blocs ou partitions triées selon la taille. Le premier élément de la liste, pointe sur la plus petite partition

1.4.2.3.3 Stratégie du pire qui convient (Worst – fit)

On alloue ***la plus grande*** zone libre. La recherche est effectuée dans toute la liste, à moins que la liste soit ordonnée par ordre ***décroissant des tailles des partitions libres***.

Exemple précédent :



Avantage : cette stratégie tente de **réduire la fragmentation** de la MC en plus petits résidus. En effet, en choisissant la plus large partition, le résidu sera assez grand pour pouvoir contenir de nouveaux programmes.

Inconvénient : cette stratégie est plus lourde à mettre en œuvre, car elle nécessitent soit de rechercher dans toutes la liste des partitions libres ordonnées (même pour Best Fit).

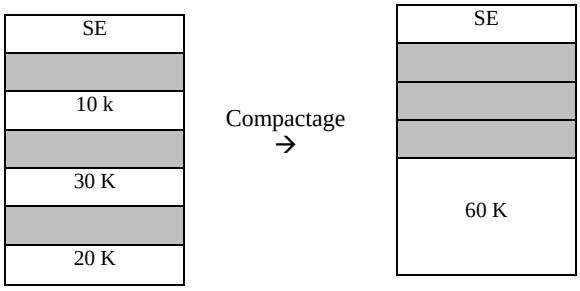
Remarque : les simulations (KNUTH 73) de ces algorithmes ont montré que Worst fit est meilleure que Best fit, et les deux sont meilleures que le First Fit.

1.4.3 Fragmentation et compactage

Dans l'allocation avec partitions variables, le phénomène le plus gênant est celui de la *fragmentation externe* de la mémoire, qui se manifeste au bout d'un certain temps de fonctionnement. Il peut, alors, arriver que l'allocateur ne puisse pas trouver une zone de taille suffisante. Une solution consiste à ***compacter*** les zones allouées en les déplaçant, toutes, vers une extrémité de la mémoire (en bas ou en haut). Ce qui fait apparaître de l'autre bout une zone libre unique dont la taille est la somme des tailles des zones libres existantes. Le compactage nécessite beaucoup de temps processeur : ***il n'est pas très recommandé***.

Exemple :

On désire exécuter le programme A de taille 35Ko,
mais aucune partition mémoire de taille supérieure
ou égale n'est disponible dans le schéma mémoire suivant :



Solution : on exécute une opération de ***compactage*** de MC.

2 Gestion de la mémoire virtuelle

Le principe de la mémoire virtuelle tend à étendre l'espace physique mémoire à un espace mémoire beaucoup plus grand, et ce, en ***utilisant une partie du disque dur*** (mémoire secondaire MS).

La mémoire virtuelle est une technique permettant d'exécuter des programmes dont la taille excède la taille de la mémoire réelle. La mémoire virtuelle permet :

- d'augmenter le nombre de processus présents simultanément en mémoire centrale
- de mettre en place des mécanismes de protection mémoire
- de partager la mémoire entre processus

2.1 Adresses logiques et adresses physiques

Les adresses générées par le *processeur* sont appelées *adresses logiques*(ou adresses virtuelles) tandis que les adresses vues par l'unité de gestion mémoire (MMU) sont dites adresses physiques(ou adresses réelles.)

2.2 Espace d'adressage logique et espace d'adressage physique

On appelle espace **d'adressage logique** (virtuel) d'un processus l'ensemble de toutes les adresses logiques (virtuelles) générées au cours de l'exécution de ce processus.

L'ensemble de toutes les adresses physique correspondant à ces adresses est appelé espace **d'adressage physique**.

Sur une machine sans mémoire virtuelle, une adresse est directement placée sur le bus de la mémoire et provoque la lecture ou l'écriture du mot référencé par l'adresse virtuelle. Dans ce cas, il n'y a pas de différence entre adresse virtuelle et réelle.

Lorsque la mémoire virtuelle est utilisée, les adresses virtuelles ne sont pas directement placées sur le bus de la mémoire. Elles sont envoyées à l'unité de gestion mémoire (MMU) ; ce composant matériel traduit les adresses virtuelles en adresses réelles (physique). Le MMU envoie des adresses réelles au bus.

Après recherche et décodage d'une instruction, le processeur envoie une adresse virtuelle au MMU qui traduit l'adresse virtuelle en adresse physique et envoie cette dernière (adresse réelle) au bus.

2.3 Le concept de mémoire virtuelle

Le concept de mémoire virtuelle se distingue des autres techniques de gestion mémoire par les aspects suivants :

1. La séparation (dissociation) de l'adressage virtuel (logique) et l'adressage physique (réel).
2. Le découpage (fractionnement) de l'espace d'adressage virtuel d'un processus en blocs (unités ou modules).
3. L'allocation non contiguë de l'espace physique.
4. L'allocation partielle de l'espace physique à un processus. C'est-à-dire permettre le chargement partiel de l'espace d'adressage logique dans l'espace physique (chargement partiel du programme d'un processus).

La **séparation** de l'espace d'adressage virtuel, de l'espace d'adressage physique est la **clé** du **concept de mémoire virtuelle.**

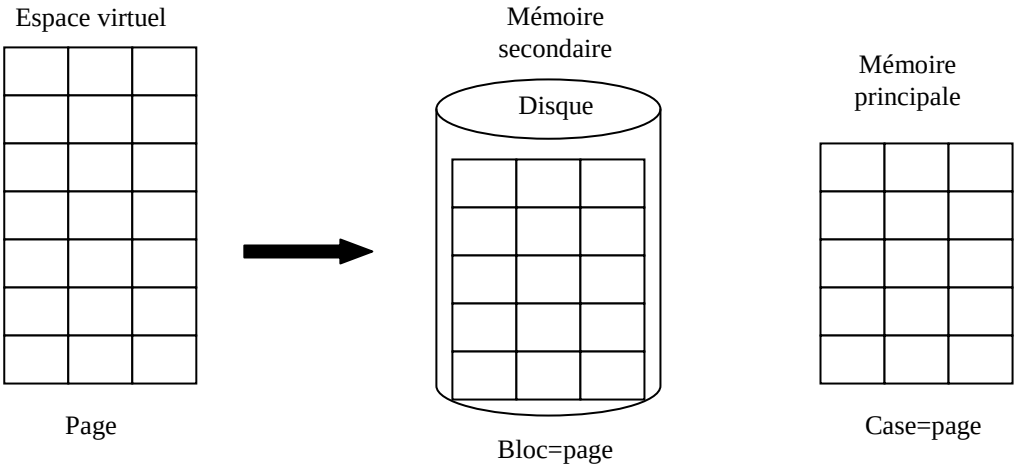
L'espace d'adressage virtuel est indépendant de l'espace d'adressage physique de la machine. Un processus peut adresser un espace d'adressage virtuel de taille beaucoup plus grande que celle de la mémoire physique présente dans le système.

2.4 La pagination

La mémoire virtuelle peut être mis en œuvre (ou implantée) de nombreuses manières, les principaux mécanismes sont : la **pagination**, la **segmentation** et la **pagination avec segmentation**.

La pagination consiste à diviser l'espace d'adressage logique en unités ou blocs appelés ***pages*** ; la mémoire physique est également découpée en blocs ou unités de même taille que les pages, appelées cases ou ***pages réelles*** (pages frames).

Les pages, des processus (espaces virtuels), qui ne sont pas chargées dans des cases de mémoire réelle sont stockées en mémoire secondaire (disque). La partie du disque (mémoire secondaire) réservée à la pagination est découpée en blocs de même taille que les pages (certains systèmes charge toutes les pages du programme en mémoire secondaire).



2.4.1 Traduction des adresses virtuelles en adresses réelles

Il est à rappeler que la traduction d'une adresse virtuelle en adresse réelle est réalisée par un dispositif matériel appelé « unité de gestion mémoire » (MMU).

1. **L'adresse virtuelle :** dans un système à mémoire paginée l'adresse est composée de :
   - Un numéro de page virtuelle (p bits de poids forts : gauche),
   - Un déplacement à l'intérieur de la page (d bits de poids faibles : droite).

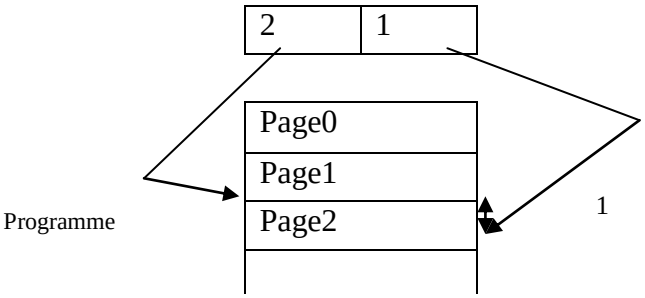
   L'adresse A=(p,d) telle que :
   P=A/taille de page ;
   d= A modulo taille de page

   Adresse virtuelle A : T : taille de la page

Page : **p** bits	Déplacement **d** bits

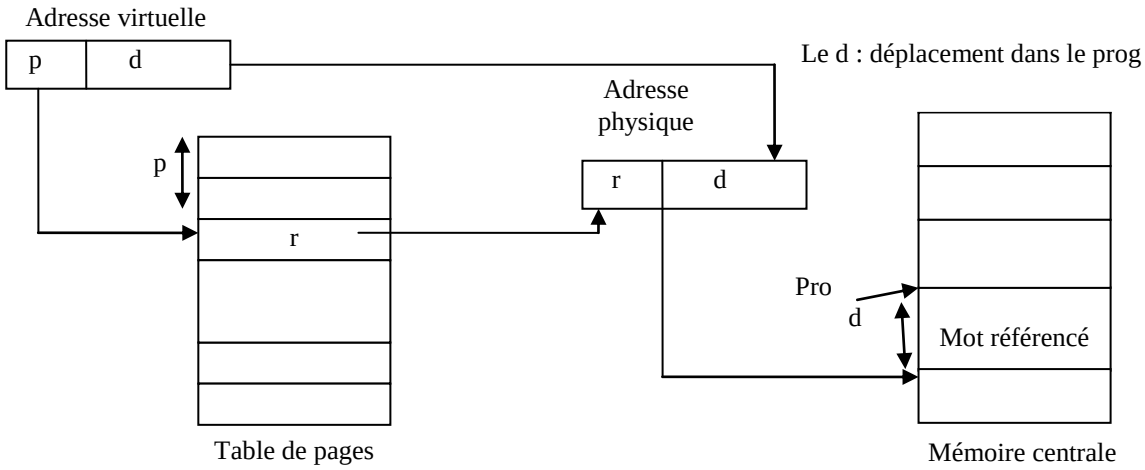
   Taille de la page = 2^{d}

Exemple : soit un mot d'adresse virtuelle 2001=(2,1) dans un système paginé de taille 1000 octets.

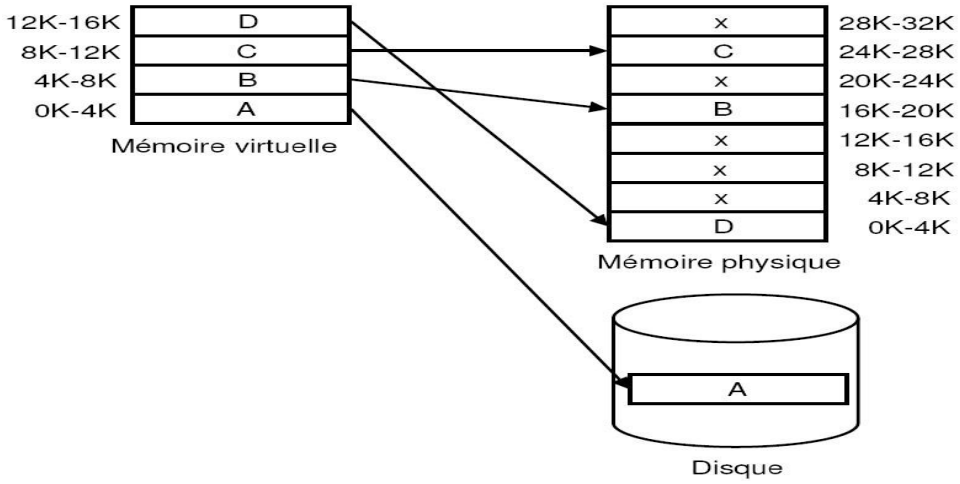


2. **Table des pages :** pour établir la correspondance entre page virtuelle et page réelle (ou case) on utilise une table que l'on appelle ***table de pages***. Cette table a **p** entrées chacune de ces entrées contient le numéro de case (adresse physique) de la page correspondante (et d'autres informations). le numéro de page **p** sert d'index dans la table des pages.

3. **Calcul de l'adresse réelle :** ***l'adresse réelle*** notée **[r, d]** d'un mot ***d'adresse virtuelle*** notée **[p, d]** est obtenue en remplaçant le numéro de page **p** par le numéro de case **r** trouvé dans la **p**^{ème} entrée de la table de pages.



Chaque processus a sa propre table de pages. L'adresse de cette table fait partie du PCB du processus. A chaque commutation de contexte, la table de pages de la machine est chargée avec la table de pages du processus élu. Si la table de pages est en mémoire centrale, l'adresse de la table de pages est chargée dans un registre (pointeur).



La ***table des pages virtuelles*** (TPV) est composée de plusieurs champs : le ***numéro de cadre*** ; un ***bit de présence*** ; un ***bit de modification (M)*** ; un ***bit de référence (R)*** ; un ***bit de protection***

2.4.2 Taille de la table de pages

La mémoire virtuelle fournit un espace d'adressage virtuel plus grand que l'espace réel ; cet espace est limité par le nombre de bits de l'adresse. La **taille** de la **table de pages** dépend de **l'espace d'adressage virtuel** et de la **taille** de

la **page**. Comme l'espace virtuel est divisé en pages, on peut dire que la taille de la table de pages est proportionnelle au nombre de pages de cet espace.

Un processus utilise rarement tout son espace virtuel. Si la table de page décrit tout l'espace virtuel, une partie de cette table ne sera pas utilisée : Pas d'espace en mémoire principale. Pour distinguer les entrées, de la table, qui sont utilisées par un processus des entrées non utilisées on utilise un bit que l'on appelle « ***bit de validité*** ». Ce bit permet de savoir si une adresse appartient on non à l'espace d'adressage d'un processus. Chaque entrée de la table de pages a un « bit de validité ».

Table de pages

	Bit de validité	Numéro de case	
Page 0	1		
Page 1	1		Pages valides : bit = 1 Pages utilisées par le processus
	1		
	1		
	0		Entrées non valides : bit = 0 Espace non utilisé par le processus
	0		
Page n	0		

2.4.3 Le choix de taille de page

Le choix de la taille de page est un paramètre souvent choisi par les concepteurs de système d'exploitation. La taille choisie peut être différente de celle qui existe sur le matériel.

Pour le choix de la taille de page optimale on doit tenir compte des paramètres suivants :

1. **Fragmentation interne :** la pagination permet de pallier le problème de la fragmentation interne ; chaque case libre peut être allouée à un processus qui la demande. Cependant, l'allocation par page fait apparaître de nouveau la fragmentation interne : La dernière page d'un espace virtuel est, en moyenne, utilisée à moitié, mais occupe une case entière.

   Plus les pages sont petites, plus la fragmentation interne est réduite.
2. **Taille de la table des pages :** pour réduire la taille de la table des pages, on doit utiliser des pages de taille assez grande. Lorsque cette table est de petite taille, on peut l'implanter à l'aide de registres rapides.

Note : On peut constater que ces 2 paramètres sont contradictoires.

Remarque : le gestionnaire de la mémoire paginée élimine le problème de la fragmentation externe. Cependant le problème de la fragmentation interne demeure entier. Si un job nécessite n pages et 1 octet, on doit lui allouer (n+1) pages physique.

2.5 La segmentation

Jusqu'ici on a considéré l'espace d'adressage virtuel comme un espace linéaire, ayant des adresses comprises entre 0 et n-1. L'utilisateur voit un programme comme un ensemble de modules de taille variable. Un module peut être une procédure ou un ensemble de données. Chaque module est défini par un **nom**.

La segmentation est une technique de gestion mémoire permettant de supporter cette ***vue de l'utilisateur***. L'espace virtuel est constitué d'un ensemble de **segments**.

Un segment correspond à un module du programme. Chaque segment est défini par : ***un nom, une longueur (taille).***

Une adresse virtuelle est composée du nom du segment et d'un déplacement à l'intérieur du segment.

2.5.1 Adresse virtuelle :

Nom de segment	Déplacement

Dans un système paginé l'adresse logique est constituée d'une seule information, c'est un dispositif matériel qui décompose l'adresse en page et déplacement. Par contre, dans un système segmenté, l'utilisateur désigne un objet dans son programme à l'aide de 2 informations : nom de segment et déplacement.

Pour simplifier l'implémentation, les segments sont numérotés et référencés par numéro plutôt que par un nom : numéro de segment, déplacement.

La numérotation des segments est réalisée à l'assemblage ou à la compilation.

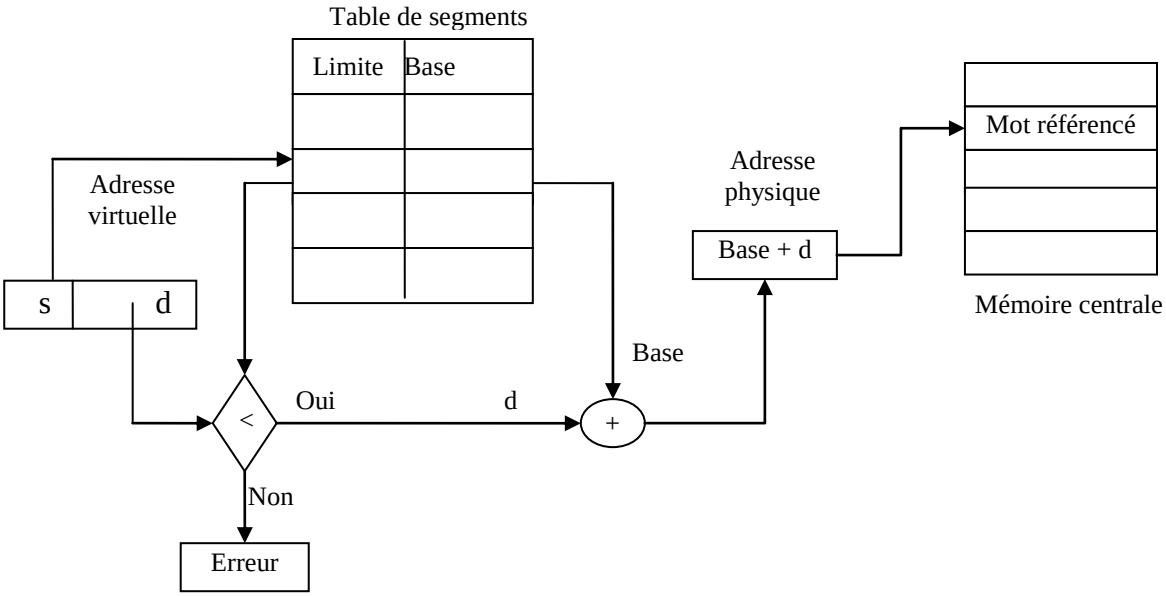
2.5.2 Dispositif de traduction des adresses

L'espace virtuel d'un processus est décrit à l'aide d'une table de segments ; chaque entrée contient :

- La base ou l'adresse début du segment en mémoire centrale,
- La limite ou longueur du segment.

La table contient autant d'entrées que de segments réellement utilisés.

Le bloc de contrôle de processus (PCB) contient un pointeur vers la table de segments.



2.6 Segmentation avec pagination

La segmentation présente 2 inconvénients majeurs :

- La fragmentation externe
- Temps de recherche d'un bloc mémoire libre pour un segment.

La solution à ces 2 problèmes consiste à paginer les segments. L'espace d'adressage virtuel est constitué d'un ensemble de segments, chacun d'eux est composé d'un certain nombre de pages.

Note : les segments sont de longueur variable, donc ils n'ont pas le même nombre de pages.

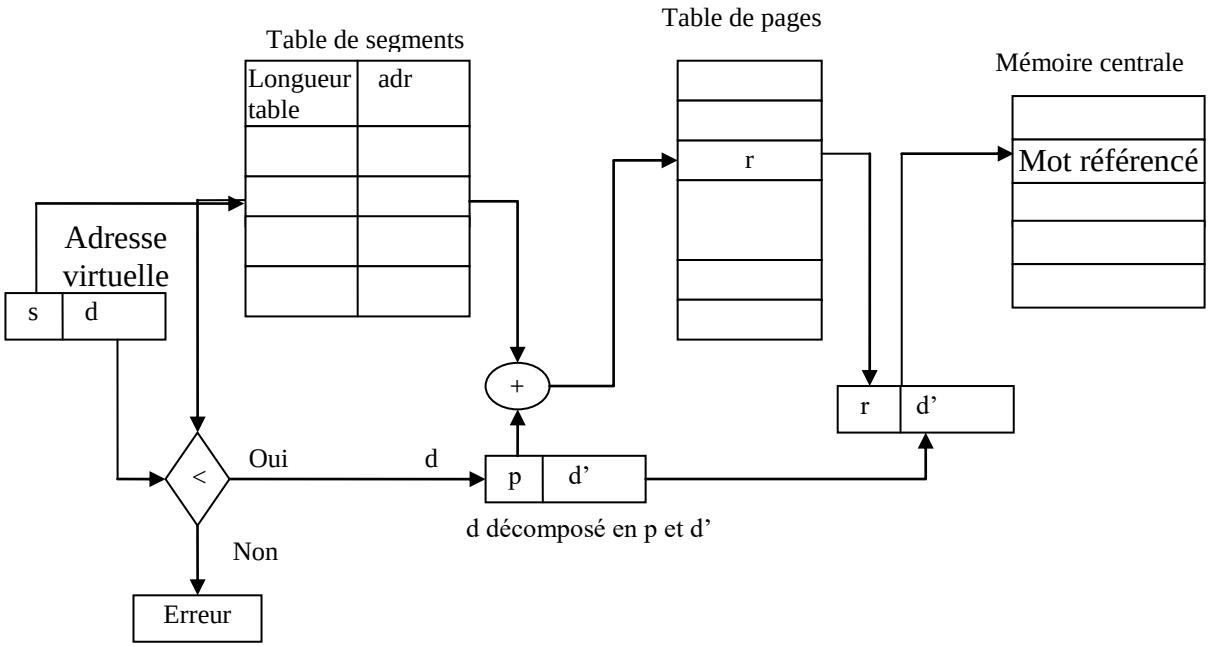
La pagination d'un segment consiste à découper le déplacement en 2 informations :

- numéro de page,
- déplacement à l'intérieur de la page.

2.6.1 Adresse virtuelle :

Numéro de segment	Déplacement dans le segment	
	Page	Déplacement

2.6.2 Traduction d'une adresse virtuelle en adresse réelle :



3 La pagination à la demande

La mémoire virtuelle offre à l'utilisateur un espace d'adressage logique (virtuel) plus grand que l'espace physique (réel). Elle permet également d'exécuter un processus sans que son programme soit entièrement chargé en mémoire centrale. Ces facilités sont réalisées à l'aide de la « **pagination à la demande** ».

Un système de pagination à la demande est caractérisé par les politiques ou stratégies suivantes :

- **La stratégie de chargement :** quand le système charge – t – il les pages en MC ?
  - Lorsque l'on a besoin : **chargement à la demande.**
  - Avant d'en avoir besoin : **pré-chargement**. Dans ce cas, on risque de charger des pages inutiles.
- **La stratégie de placement :** où le système charge – t – il les pages en MC ?

  Toutes les cases de la MC sont équivalentes, donc la politique de placement n'a pas de conséquences sur les performances du système de pagination.
- **La stratégie de remplacement :** comment le système choisit – t – il les pages qui doivent être retirées de la MC, lorsque aucune case n'est plus disponible ?

  Pour le choix d'une page, on peut tenir compte des informations sur l'utilisation passée des pages présentes en MC

Dans un système de pagination à la demande une page n'est chargée en MC que si elle a été référencée. On peut donc, lancer l'exécution d'un processus sans qu'aucune page de son programme ne soit chargée en MC. Un processus peut alors référencer un mot se trouvant dans une page qui n'est pas encore chargée en MC.

La table de pages est utilisée directement par le mécanisme de traduction des adresses (MMU). Une entrée de la table de pages contient des informations telles que :

- **Numéro de case mémoire :** correspond à la case mémoire.
- **Bits de protection :** page en lecture/ écriture, en lecture, lecture / exécution.

- **Bit de validité :** page utilisée ou non (un processus utilise une partie seulement de l'espace virtuel),
- **Bit de présence** : page chargée en MC ou non.
- **Bit de modification :** page modifiée ou non depuis le dernier chargement.
- **Bit de référence :** page référencée ou non.

Lorsqu'un défaut de page se produit, le système doit charger la page correspondante depuis la mémoire auxiliaire dans une case de la mémoire principale.

3.1 Détection de défaut de page

La solution consiste à rajouter un bit, que l'on appellera « **bit** de **présence** », par entrée de la table de pages :

- Bit = 0 ➔ page non chargée en MC : défaut de page.
- Bit = 1 ➔ page chargée en MC.

3.2 Traitement de défaut de page

Un défaut de page est détecté lors de la traduction d'une adresse virtuelle en adresse réelle, en accédant à la table de pages et en testant le bit de présence (test réalisé par le MMU).

- Bit de présence = 0 ➔ prendre le numéro de case contenu dans cette entrée.
- Bit de présence = 1 ➔ la page n'est pas présente en MC, donc **déroutement pour défaut de page**.

Procédure de traitement de défaut de page :

1. Sauvegarder le contexte du processus interrompu,
2. Allouer une case à la page et mettre à jour la table d'allocation : si pas de case libre alors choisir une « page victime » à l'aide d'un algorithme de remplacement.
3. Localiser la page en mémoire secondaire (MS).
4. Charger la page, à partir de la MS, dans la case allouée. Il est à noter que pendant le temps du transfert de la page, le processeur est alloué à un autre processus
5. Après le transfert de la page dans la case allouée, mettre à jour la table de pages en positionnant le bit à 1 et mettant le N° de la case dans l'entrée correspondante.
6. Restaurer le contexte du processus interrompu et **reprendre l'exécution de l'instruction** ayant provoqué le déroutement défaut de page.

3.3 Remplacement de page

Quand il y a un défaut de page, on doit allouer une case pour charger la page référencée. S'il n'existe pas de page, on choisit une parmi celles qui sont présentes en MC et on affecte sa case à la page référencée. La page sélectionnée au moyen d'un algorithme de remplacement est appelée « page victime ». En générale, un algorithme de remplacement choisit une page en tenant compte des informations sur l'utilisation passée de la page (page en cours d'utilisation par une opération E/S, page référencée ou non, page modifiée ou non).

3.3.1 Evaluation des algorithmes de remplacement

Un algorithme de remplacement est évalué en l'exécutant sur une suite ω de références mémoire (ou chaîne de références) avec un nombre de cases **m** et en calculant le nombre de défaut de page **f (ω, m)**.

3.3.2 Les algorithmes de remplacement

La politique de remplacement est l'aspect le plus critique de tout système de pagination. Il existe une multitude d'algorithmes de remplacement ; dans ce qui suit, nous étudierons les principaux algorithmes de remplacement.

3.3.2.1 L'Algorithme FIFO

C'est l'algorithme le plus simple. Quand on veut remplacer une page, on choisit la page la plus ancienne.

Chaque fois que l'on charge une page en MC, on enfile le numéro de cette page dans une file gérée en FIFO.

Exemple :

Soit un programme de 5 pages virtuelles numérotées de 1 à 5, et une MC de taille de 3 pages physique.

Voyons l'image de la MC à chaque référence mémoire de la chaine CR : chaîne de référence suivante :

4 3 2 1 4 3 5 4 3 2 1 5														
CR		4	3	2	1	4	3	5	4	3	2	1	5	Total DP
MC	P0	4	4	4	1	1	1	5	5	5	5	5	5	
	P1	libre	3	3	3	4	4	4	4	4	2	2	2	
	P2	libre	libre	2	2	2	3	3	3	3	3	1	1	
Défaut de page (DP)		1	1	1	1	1	1	1	0	0	1	1	0	9

P0 à p2 : les cadres de pages physique en MC. Taux de défauts de page f=nombre DP/taille de la CR=9/12=75%

3.3.2.2 L'Algorithme optimal (OPT ou MIN)

Un algorithme de remplacement optimal minimise le taux de défauts de page. Cet algorithme choisit la page qui ne sera plus utilisée ou la page qui sera utilisée le plus tard possible. Cet algorithme suppose une connaissance de l'ensemble de la chaîne de référence ω ; il est donc irréalisable en temps réel. Il permet d'évaluer, par comparaison, les autres algorithmes.

3.3.2.3 L'Algorithme LRU (Least Recently Used)

L'algorithme LRU (moins récemment utilisé) choisit la page qui n'a pas été utilisée depuis longtemps (ou la page ayant fait l'objet d'une référence la plus tardive). Cette classe d'algorithme est appelée algorithme à pile.

3.3.2.4 L'Algorithme NRU (Not Recently Used; non récemment utilisé)

A chaque entrée de la table de page, on associe :

- Un bit de référence R ;
- Un bit de modification M.

Au lancement (démarrage) du processus tous les bit R et M de la table de page sont initialisés à zéro. A chaque accès en lecture à une page, R est mis à 1. A chaque accès en écriture, M est mis à 1. Pour distinguer les pages qui n'ont pas fait l'objet d'une référence récente des autres pages, les bits R sont remis à zéro à chaque top d'horloge.

Lorsqu'un défaut de page se produit, on parcourt toutes les pages et on les réparti en quatre classes en fonction des valeurs de bits R et M.

Classes	**R**	**M**	
Classe 0	0	0	Page non référencée et non modifiée
Classe 1	0	1	Page non référencée et modifiée
Classe 2	1	0	Page référencée et non modifiée
Classe 3	1	1	Page référencée et modifiée

L'algorithme NRU choisit une page au hasard de la classe non vide qui a le plus petit numéro (0, 1, 2, 3).

Conclusion

Les performances du SE en dépendent de la gestion de la MC qui est accomplie par un module du SE appelé gestionnaire de la mémoire centrale MMU. Ce dernier doit gérer l'espace mémoire alloué aux process en terme d'allocation et restitution, mais aussi en terme de protection et partage des informations.

Plusieurs stratégies de gestion de la MC dans un système multiprogrammé ont été développées qui peuvent être regroupées en deux grandes approches :

- Approche allocation contiguë
- Approche d'allocation non contiguë.

La première considère l'espace d'adressage d'un programme comme étant une seule entité de mots insécable (inséparable). On trouve les techniques d'allocation par partitions statiques et variables.

La deuxième approche voie un programme comme étant une entité de mots sécable (séparable) en un ensemble de morceaux insécables. Ainsi chaque morceau peut être logé dans une zone suffisante uniquement à lui indépendamment du reste du programme. On en trouve la segmentation, la pagination et la segmentation paginée. La première considère les tailles des morceaux non identiques, tant que la pagination exige que les morceaux aient une taille identique et fixe.

La mémoire virtuelle propose deux avantages :

1- un programme d'une taille supérieure à celle de la MC peut être chargé et exécuté dans la MC.

2- un programme peut être exécuté malgré qu'il est partiellement chargé en MC.

La MV crée une indépendance entre l'espace d'adressage logique tel qu'il est vu par l'utilisateur et l'espace d'adressage physique.

La pagination à la demande est la forme d'implémentation la plus répandue de la MV. Elle se base sur le principe qu'une page se situe par défaut dans la MV et elle n'est chargée en MC qu'au moment ou elle est référencée par le processus actif donnant lieu un déroutement (défaut de page). Pour traiter le déroutement, un remplacement de pages est effectué par le choix d'une page victime ; plusieurs stratégies sont proposées, leurs points commun est d'analyser la chaîne de référence dans le passé pour essayer de prévoir le comportement du processus dans un future.