Université Mohamed Kheider Biskra
Faculté des sciences exactes et sciences de la nature et de la vie
Département d'informatique

Support de cours

Module : Systèmes d'exploitation I

Chapitre 2 : Gestion de la mémoire

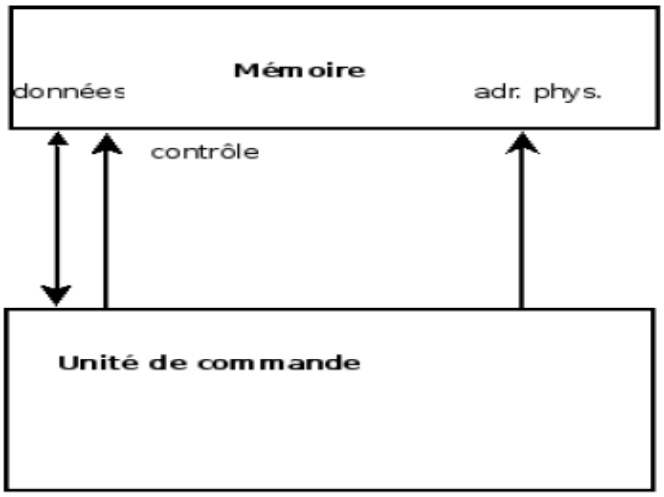
(Partie 01)

Niveau : 2LMD
Chargé de module : Mme. D. Boukhlouf

Année universitaire 2017/2018

1. Introduction

La mémoire est un organe qui a pour fonction de stocker et restituer des mots binaires repérés par une adresse. Elle communique avec le reste de l'ordinateur par 3 bus :
- le bus de contrôle (bus de commande), qui indique à la mémoire l'opération que l'on veut effectuer
- le bus de données, bidirectionnel, qui sert à émettre et recevoir les mots ;
- le bus d'adresses, qui indique à la mémoire l'adresse concernée.



La mémoire est une ressource importante qui doit être gérée avec attention. Elle est capable de stocker :

- Des objets passifs comme répertoires des fichiers (des programmes, des images, de textes et de sons), des tampons de données d'entrées/ sorties ou de transfert de réseau.
- Des objets actifs comme les processus.

Elle est divisée en une suite finie de composants appelés emplacements. Un programme ne peut pas s'exécuter que si ses instructions et ses données (au moins partiellement) sont en mémoire centrale. Si l'on désire exécuter plusieurs programmes simultanément ; il faut que chacun soit chargé dans la mémoire principale.

2. Le système de gestion de la mémoire (*Memory manager*): Un gestionnaire de la mémoire doit assurer les fonctions suivantes :

1. garder trace du statut de chaque portion de la mémoire (libre ou allouée).
2. déterminer une politique d'allocation de la mémoire qui permet de décider qui doit avoir de la mémoire, combien et pour combien du temps.
3. déterminer une technique d'allocation qui permet d'allouer de l'espace à un processus.
4. déterminer une politique de libération mémoire.

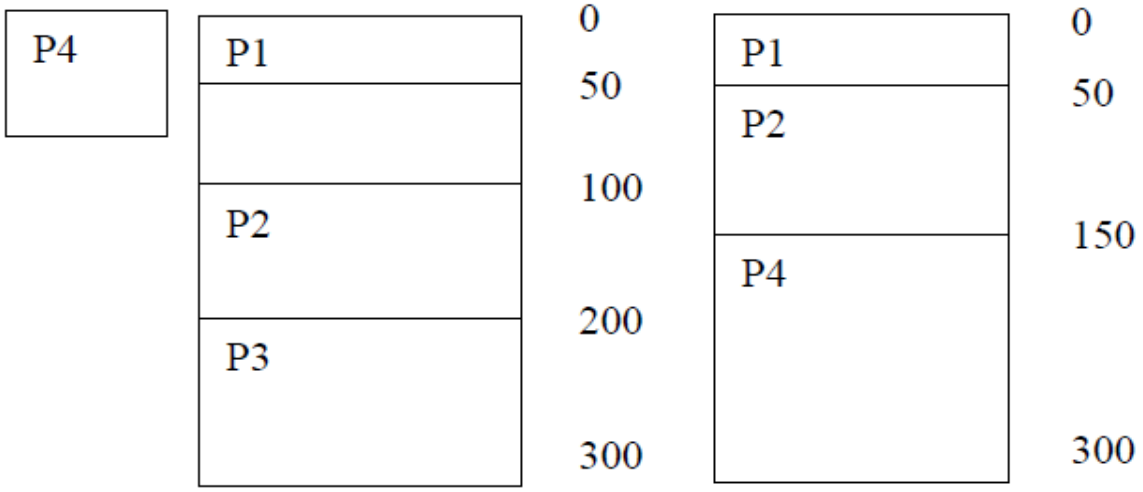
Le but d'une bonne gestion de la mémoire est d'augmenter le rendement global du système. Pour cela, il faut qu'il y'ait suffisamment de processus (taches) en mémoire pour assurer une bonne répartition entre les entrées/sorties et la demande de CPU. Il doit viser les objectifs suivants :

2.1 La réallocation : le système d'exploitation alloue à chaque programme une zone mémoire. Mais tous les programmes n'ont pas la même taille. De plus, les programmes sont lancés par les utilisateurs, puis terminent à ces moments que le système ne connaît pas à l'avance. Chaque fois qu'un utilisateur demande le lancement d'un programme, le système doit trouver une place dans la mémoire pour le charger : il y'a réorganisation des programmes en mémoires.

Exemple :

Soient les programmes P1,P2 et P3 en mémoire selon la figure suivante. Si un quatrième programme de taille 150 demandait à être exécuté, ce serait évidemment impossible. Maintenant, supposons que le programme P3 se termine, la place totale disponible dans la mémoire est de 150. le programme P4 peut donc théoriquement être chargé. Malheureusement, dans notre exemple, ce sera encore impossible, car les 150

unités disponibles ne forment pas un espace contigu dans lequel on peut charger le programme P4. Un réarrangement de l'espace mémoire permettra de charger P4 comme suit :



2.2 La protection : la coexistence de plusieurs processus en mémoire centrale nécessite la protection de chaque espace mémoire vis à vis des autres. Par conséquent, un processus P1 ne peut accéder à l'espace d'un processus P2 que s'il est autorisé.

2.3 Le partage : parfois, il est utile de partager un espace mémoire entre plusieurs processus. Ainsi, le sous –système de gestion de la mémoire doit autoriser des accès contrôlés sans compromettre la protection.

Exemple : Un éditeur de texte partagé entre plusieurs utilisateurs d'un système à temps partagé.

3. Stratégies d'allocation de la mémoire :

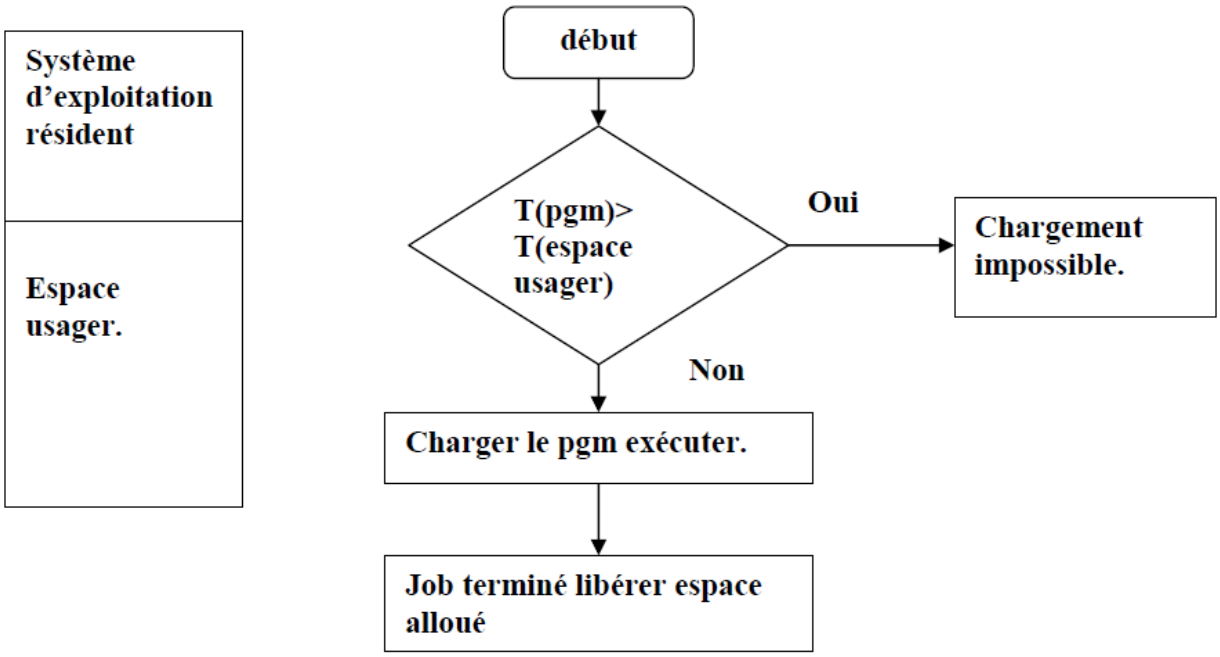
On recense essentiellement deux modes de gestion de la mémoire centrale : le mode contigu et celui non contigu. Selon le mode de gestion de la mémoire qui est appliqué, lorsqu'un programme est chargé en mémoire centrale à partir du disque, le programme sera placé dans une seule zone (allocation contiguë) ou réparti entre plusieurs zones (allocation non contiguë).

3.1 Allocation contiguë

3.1.1 Une seule zone contiguë (monoprogrammation) :

Dans ce cas, la mémoire est divisée en deux zones contiguës. L'une est allouée en permanence à la partie résidente du système d'exploitation. La deuxième zone permettra de reloger un processus usager ou du système non résident. Cette stratégie est utilisée par les micro-ordinateurs sous PC/DOS.

La gestion de la mémoire s'avère simple, le système d'exploitation doit garder trace de deux zones mémoires. L'algorithme d'allocation peut être décrit par l'organigramme ci-contre : l'inconvénient majeur de cette stratégie est la mauvaise utilisation de l'espace mémoire dans le sens où les programmes occupent (en général) partiellement l'espace usager. Par ailleurs, la taille des programmes usagers est limitée par la taille de l'espace usager.

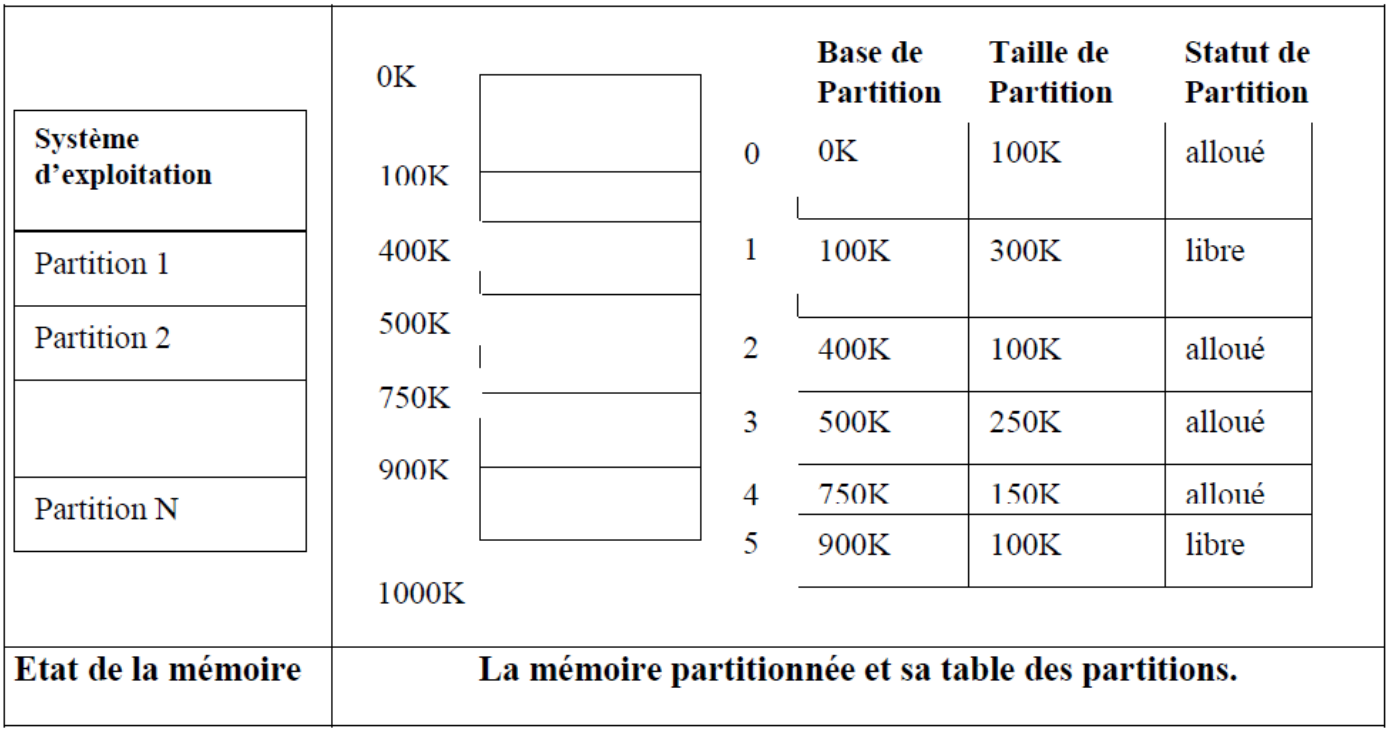


3.1.2 Partitions multiples (multiprogrammation)

La plupart des SE modernes autorisent l'exécution de processus multiples en même temps: Lorsqu' un processus est bloqué en attente d'une E/S, un autre peut utiliser la CPU.

3.1.2.1 Partitions multiples statiques

La manière la plus simple de faire de la multiprogrammation consiste à subdiviser la mémoire en n partitions de taille fixe. Le nombre et la taille de chaque partition sont fixés à la génération du système. Chaque partition peut contenir exactement un seul processus. Le gestionnaire de la mémoire maintient une table indiquant les parties de la mémoire disponibles et celles qui sont occupées : une table de description des partitions (Partition Description Table : PDT)..



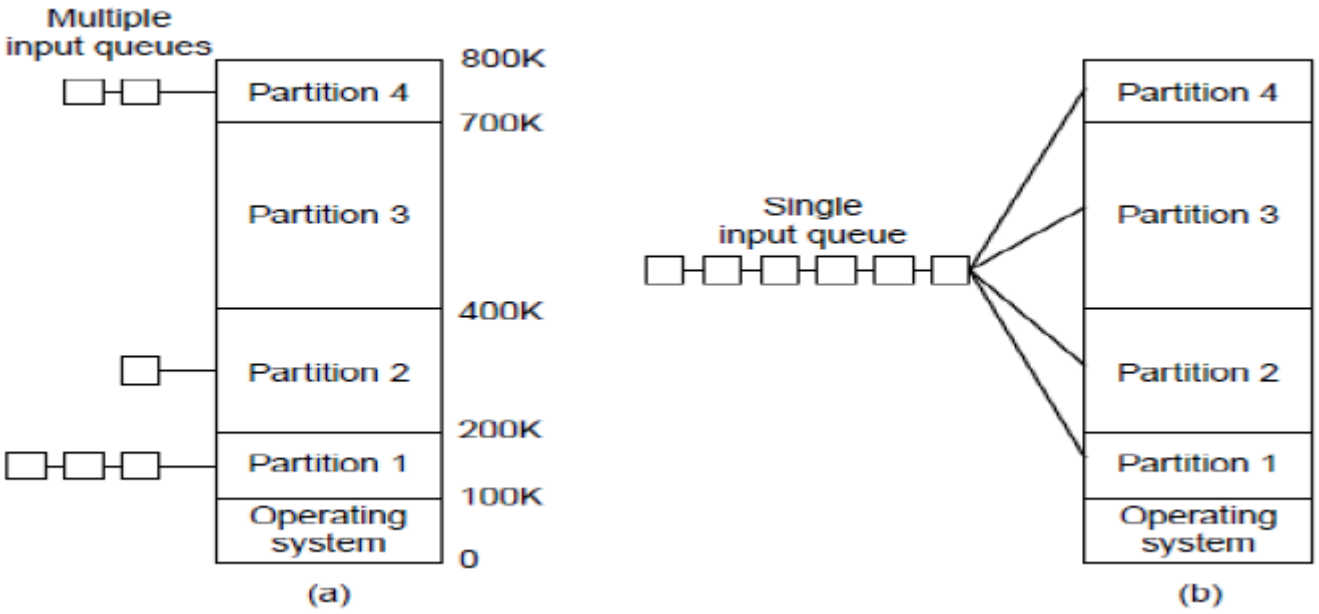
	Base de Partition	Taille de Partition	Statut de Partition
0	0K	100K	alloué
1	100K	300K	libre
2	400K	100K	alloué
3	500K	250K	alloué
4	750K	150K	alloué
5	900K	100K	libre

Etat de la mémoire — **La mémoire partitionnée et sa table des partitions.**

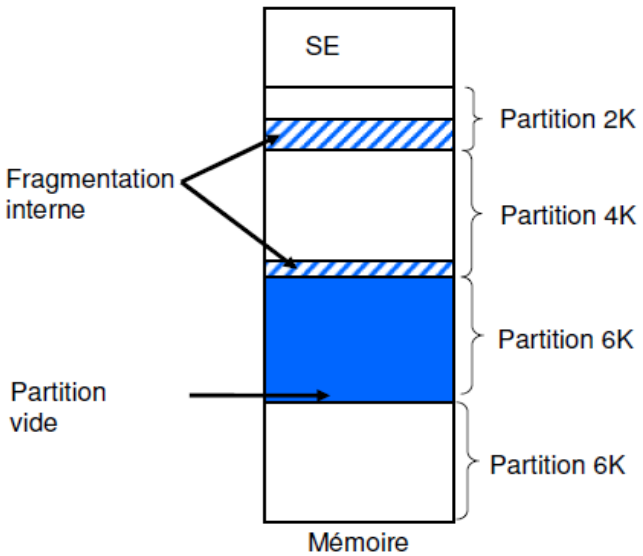
Il existe deux méthodes de gestion :

- on crée une file d'attente par partition . Chaque nouveau processus est placé dans la filed'attente de la plus petite partition pouvant le contenir.

Inconvénients :
* on perd en général de la place au sein de chaque partition
* il peut y avoir des partitions inutilisées (leur file d'attente est vide)
- on crée une seule file d'attente globale. Il existe deux stratégies :
* dès qu'une partition se libère, on lui affecte la première tâche de la file qui peut y tenir. Inconvénient : on peut ainsi affecter une partition de grande taille à une petite tâche et perdre beaucoup de place
* dès qu'une partition se libère, on lui affecte la plus grande tâche de la file qui peut y tenir. Inconvénient : on pénalise les processus de petite taille.



Fragmentation interne:
La mémoire allouée peut être plus grande que le mémoire requise. Cette différence est interne à une partition mais n'est pas utilisée.
Fragmentation externe:
Lorsqu'aucune partition n'est suffisante pour accueillir un processus, alors que la somme des partitions libers permet le chargement du processus.



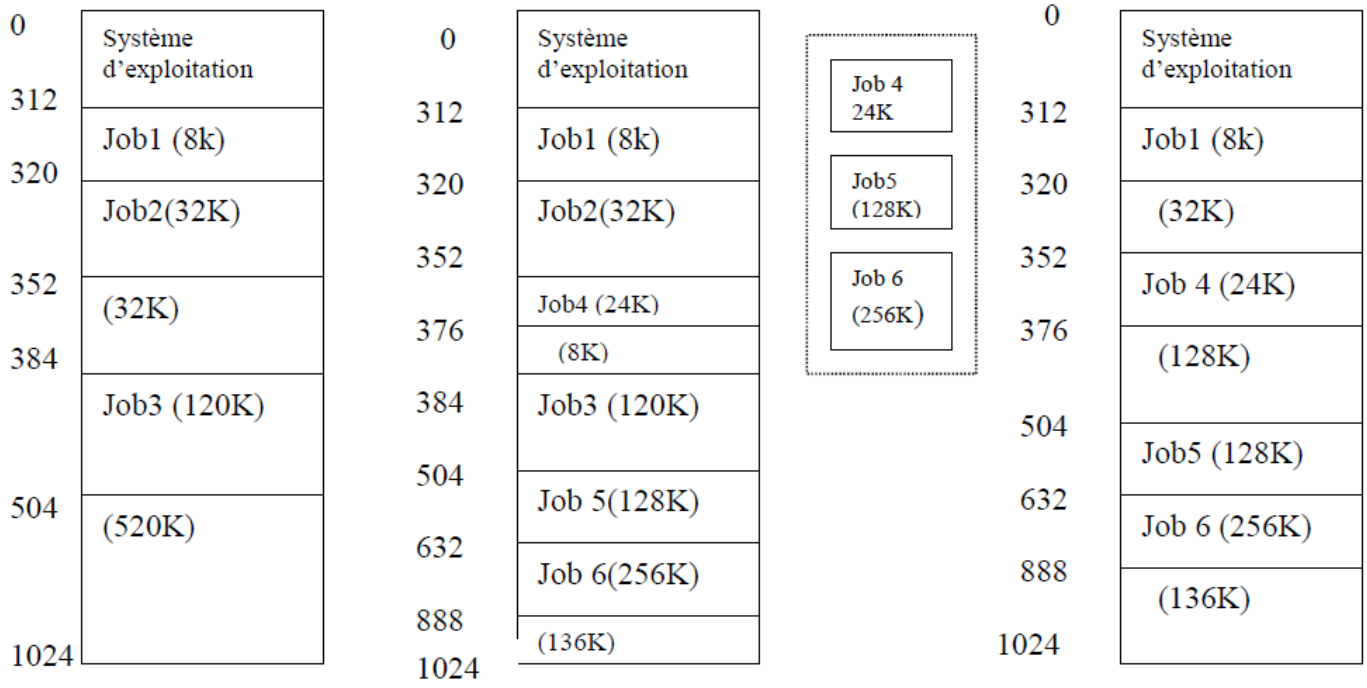
3.1.2.2 Partitions multiples variables :
Afin d'améliorer la stratégie de la gestion de la mémoire à partitions fixes, on partitionnera la mémoire dynamiquement selon la demande. A chaque programme est allouée une partition exactement égale à sa

taille. Quand un programme termine son exécution, sa partition est récupérée par le système pour être allouée à un autre programme complètement ou partiellement selon la demande.

Exemple :

Soit une mémoire de 1024 Ko dont 312 Ko sont occupés par le SE. A l'instant T1, on dispose de la configuration selon la figure ci dessous.

Dans une organisation à partitions variables, le placement des programmes dans les partitions se fait suivant différentes stratégies. Pour cela, le gestionnaire de la mémoire doit garder trace des partitions occupées et des partitions libres. Une solution consiste à maintenir deux tables dont l'une est destinée aux partitions occupées et l'autre aux partitions libres.



(a) état initial.

(b) allocation des partitions pour les jobs 4.5 et 6.

(c) job 2 et 3 terminent et libèrent leurs partitions.

Stratégies de placement:

Dans une organisation à partitions variables, le placement des programmes dans les partitions se fait selon un certain nombre de stratégies possibles. Il est à noter que certain nombre de stratégies possibles. Il est à noter que le comportement d'une stratégie dépend beaucoup de la nature des demandes :

a) **Stratégie du premier qui convient (the first fit) :** dans cette stratégie, la table des partitions libres est triée par ordre des adresses croissantes. Pour l'allocation d'une partition donnée, on commencera par la partition libre de plus basse adresse et la recherche continue jusqu'à la rencontre de la première partition dont la taille est au moins égale à celle du programme en attente.

b) **Stratégie du meilleure qui convient (the best fit) :** dans ce cas, la table des partitions libres est triée par tailles croissantes. Pour l'allocation d'une partition donnée, on commencera par la partition libre de plus petite taille. La recherche continue jusqu'à la rencontre de la première partition dont la taille est au moins égale à celle du programme en attente.

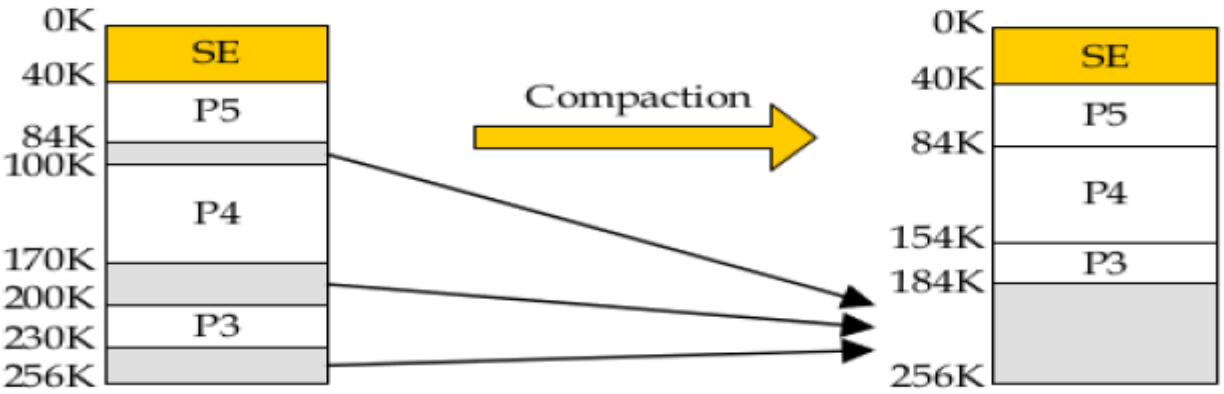
c) **Stratégie du pire qui convient (the Worst fit) :** dans ce cas, on alloue au job la partition de plus grande taille. On doit parcourir toute la table sauf s'elle est triée.

Remarque

Les deux autres stratégies sont lourdes à mettre en œuvre ; Un problème de fragmentation externe se pose et la solution c'est le compactage.

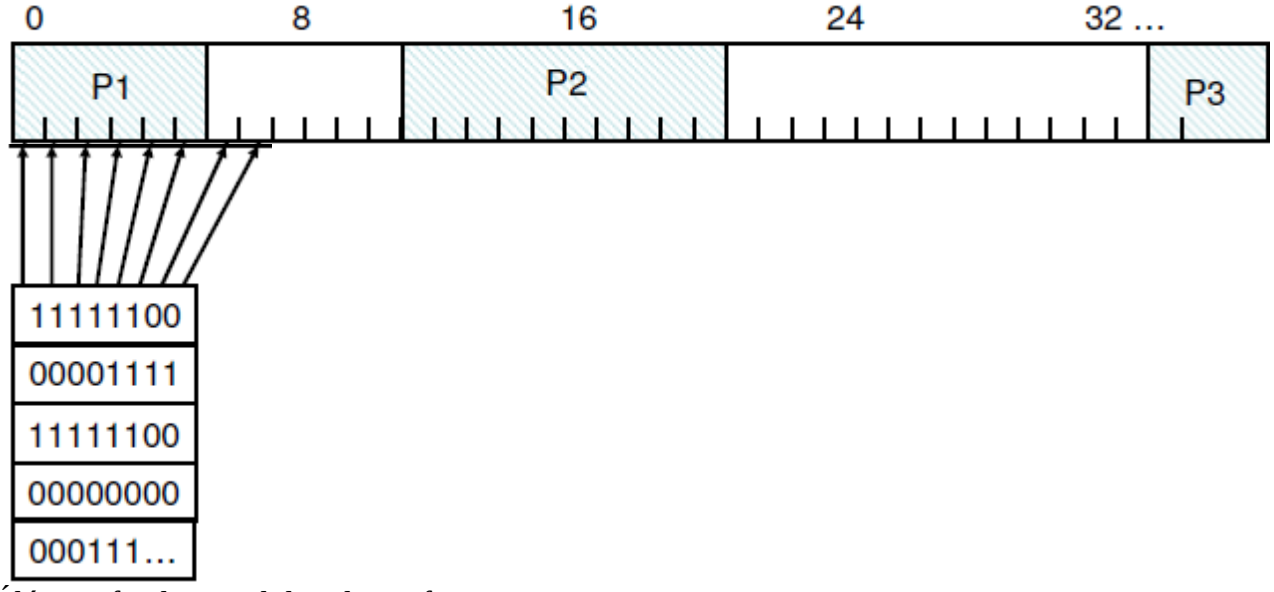
4.2.3 Le compactage :

Le compactage est une solution pour la fragmentation externe qui permet de regrouper les espaces inutilisés. L'opération de compactage est effectuée quand un programme qui demande d'être exécuté ne trouve pas une partition assez grande, mais sa taille est plus petite que la fragmentation externe existante



Pour gérer l'allocation et la libération de l'espace mémoire, le gestionnaire doit connaître l'état de la mémoire:

a) Table de bits : on peut conserver l'état des blocs de mémoire grâce à une table de bits. Les unités libres étant notées par 0 ceux occupées par un 1 (ou l'inverse).

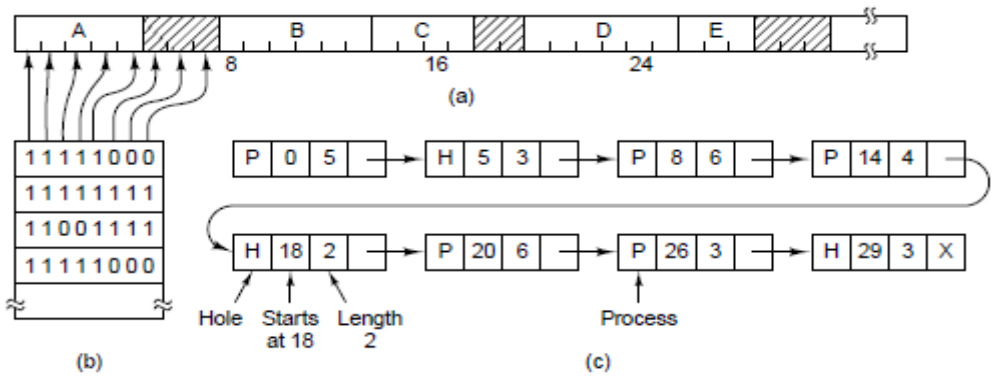


Élément fondamental dans la configuration:

Plus l'unité est petite, plus le tableau de bits est important Lorsque **l'unité d'allocation est plus grande, le tableau de bits est plus petit**, mais une quantité **non négligeable** de mémoire peut être gaspillée dans la dernière unité allouée à un processus ayant une taille qui n'est pas un multiple de l'unité d'allocation.

Avantage du *bit map*: Moyen simple de garder une trace des mots mémoire dans une quantité fixe de mémoire Inconvénient du *bit map*: Lorsqu'un processus de k unités est chargé en mémoire, le gestionnaire de mémoire doit parcourir le *bit map* pour trouver une suite de k bits consécutifs dont la valeur est 0. **Cette recherche peut être lente**.

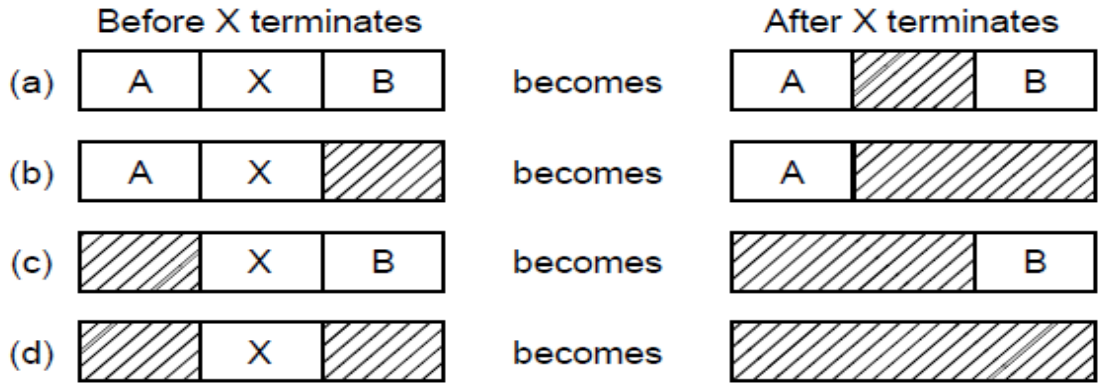
Maintenir une liste chaînée des segments de mémoire allouées et libres. Dans cette liste un élément est soit un processus, soit un trou entre deux processus.



b) Listes chaînées : on peut représenter la mémoire par une liste chaînée de structures dont les membres sont : le type (libre ou occupé), l'adresse de début, la longueur et un pointeur sur l'élément suivant. A l'achèvement d'un processus ou à son transfert sur disque, il faut du temps (mise à jour des listes chaînées) pour examiner si un regroupement avec ses voisins est possible pour éviter une fragmentation excessive de la mémoire.
En résumé, les listes chaînées sont une solution plus rapide que la précédente pour l'allocation, mais plus lente pour la libération.

c) Libération d'une partition :
Trois cas sont à considérer lors de la libération d'une partition :

- la partition libérée est entourée de deux partitions libres,
- la partition libérée est entourée d'une partition libre et d'une partition occupée,
- la partition libérée est entourée de deux partitions occupées.



4. Le va –et-vient :
Le va-et-vient est mis en oeuvre lorsque tous les processus ne peuvent pas tenir simultanément en mémoire. On doit alors en déplacer temporairement certains sur une mémoire, en général, une partie réservée du disque (swap area ou baking store).
Sur le disque, la zone de va-et-vient d'un processus peut être allouée à la demande dans la zone de va-et vient générale (swap area). Quand un processus est déchargé de la mémoire centrale, on lui recherche une place. Les places de va-et-vient sont générées de la même manière que la mémoire centrale. La zone de va-et-vient d'un processus peut aussi être allouée une fois pour toute au début de l'exécution. Lors du déchargement, le processus est sur d'avoir une zone d'attente libre sur le disque.
Le système exécute pendant un certain quantum de temps les processus en mémoire provisoire. L'algorithme de remplacement peut être le tourniquet.

Espace d'adressage logique ou physique

L'unité centrale manipule des **adresses logiques** (emplacement relatif).Les programmes ne connaissent que des adresses logiques, ou virtuelles. L'espace d'adressage logique (virtuel) est donc un ensemble d'adresses pouvant être générées par un programme.

L'unité mémoire manipule des **adresses physiques** (emplacement mémoire). Elles ne sont jamais vues par les programmes utilisateurs. L'espace d'adressage physique est un ensemble d'adresses physiques correspondant à un espace d'adresses logiques.

La mémoire virtuelle :

La taille d'un processus doit pouvoir dépasser la taille de la mémoire physique disponible, même si l'on enlève tous les autres processus. Le principe de la **mémoire virtuelle** : le système d'exploitation conserve en mémoire centrale les parties utilisées des processus et stocke, si nécessaire, le reste sur disque. Mémoire virtuelle et multiprogrammation se complètent bien : un processus en attente d'une ressource n'est plus conservé en mémoire centrale, si cela s'avère nécessaire. La mémoire virtuelle fait appel à deux mécanismes : segmentation ou pagination. La mémoire est divisée en segments ou pages. Sans recours à la mémoire virtuelle, un processus est entièrement chargé à des adresses contiguës ; avec le recours à la mémoire virtuelle, un processus peut être chargé dans des pages ou des segments non contigus.