Gestion de la mémoire

SITE : http://www.sir.blois.univ-tours.fr/~mirian/

Introduction

- La mémoire est une ressource importante qui doit être gérée avec attention.
- Même si la quantité de mémoire d'un ordinateur a beaucoup augmentée, la taille des programmes s'accroît aussi.
- La situation idéale serait de donner à chaque programmeur une mémoire infiniment grande, infiniment rapide, non volatile et, de plus, bon marché. **La technologie ne fournit pas de telles mémoires.**
- **Hiérarchisation de la mémoire**: les ordinateurs ont une petite quantité de mémoire très rapide, chère et volatile (mémoire RAM) et beaucoup de gigabytes de mémoire plus lente, bon marché et non volatile.
- Le SE a le rôle de coordonner l'utilisation des différentes mémoires.

La mémoire et le gestionnaire de la mémoire

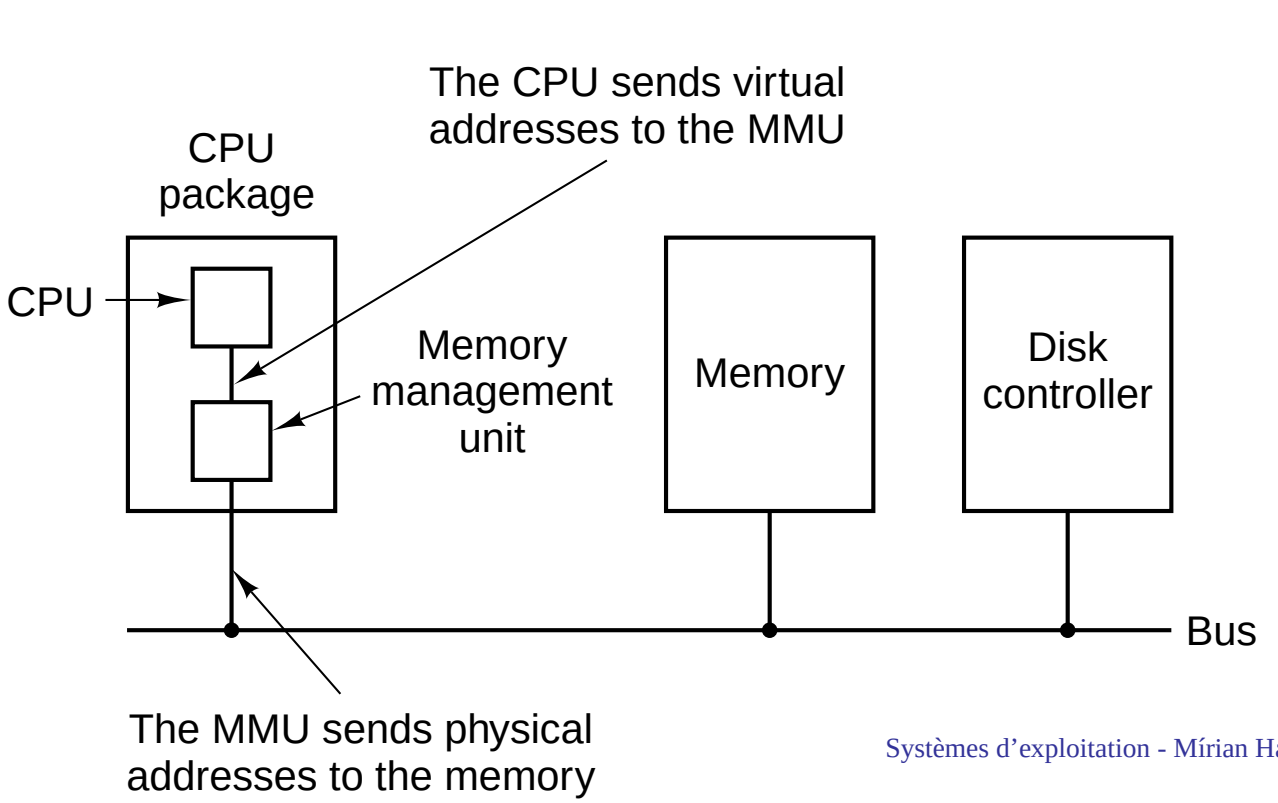
- **Mémoire**: grand **tableau** de mots (octets), chacun possédant sa propre adresse.
- La CPU extrait les instructions de la mémoire en fonction de la valeur d'un compteur d'instructions.
- Système de gestion de la mémoire (*Memory manager*): partie du SE qui gère la hiérarchie de stockage
  - Suivre les parties de la mémoire qui sont utilisées ou non utilisées.
  - Allouer/libérer espace mémoire aux processus.
  - Contrôler le *swapping* entre la mémoire principale et le disque.

Adresse logique et adresse physique

- **Adresse logique ou adresse virtuelle (*virtual address*)**
  Adresse générée par la CPU.
- **Adresse physique**
  Adresse vue par l'unité de mémoire.
- **MMU (*Memory Management Unit*)**: Dispositif matériel qui fait la conversion des adresses virtuelles à physiques

Schéma de conversion

- Registre de base = registre de translation (*realocation register*).
- La valeur du registre de translation est additionnée à chaque adresse générée par un processus utilisateur au moment où il est envoyé à la mémoire.
- Le programme utilisateur n'aperçoit jamais les adresses physiques; il traite les adresses logiques.



Types basic de gestion

Les systèmes de gestion de la mémoire peuvent être partagés en deux groupes:

- Ceux qui déplacent un processus entre le disque et la mémoire pendant sont exécution.
- Ceux qui ne font pas cela (plus simple).

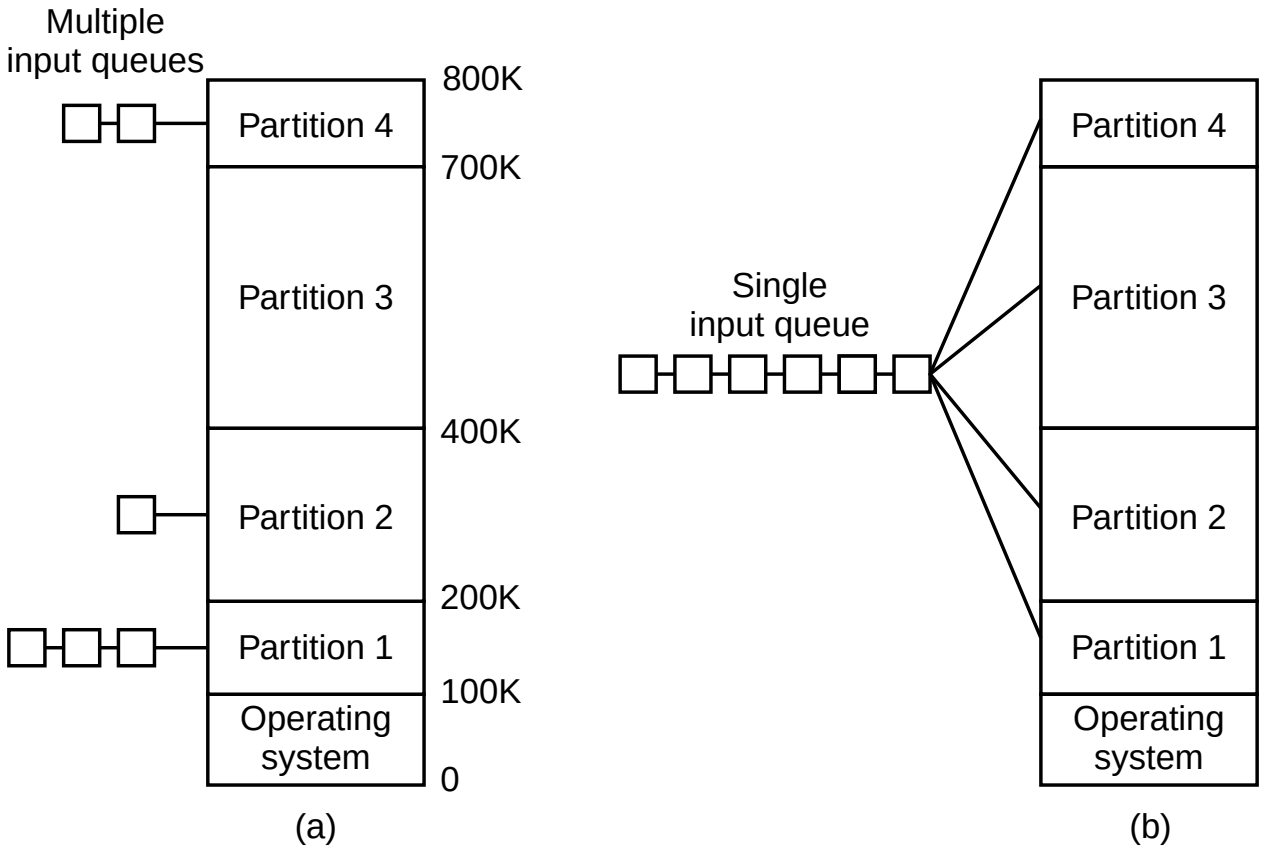
Monoprogrammation

- La manière la plus simple de gestion de mémoire: **exécuter un seul programme à la fois**.
- La mémoire est habituellement **subdivisée en deux partitions**, une pour le SE résident (en général en mémoire basse) et l'autre pour les processus utilisateur (mémoire haute).
- Utilisé par certains (petits) MS-DOS. Dans les IBM-PC, la portion du système dans la ROM est appellée BIOS (Basic Input Output System).

Multiprogrammation avec partitions fixes

- 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.**
- La manière la plus simple de faire de la multiprogrammation consiste à **subdiviser la mémoire en n partitions de taille fixe**. **Chaque partition peut contenir exactement 1 processus**.
- Degré de multi programmation = nombre de partitions.
- SE maintient une table indiquant les parties de la mémoire disponibles et celles qui sont occupées.
- Trou (*hole*): bloc de mémoire disponible.
- Puisque les partitions sont fixes, chaque espace inutilisé est perdu.

Multiprogrammation avec partitions fixes



- Partition fixes avec des files d'attente différentes
- Partition fixes avec une seule file d'attente différentes

Multiprogrammation: Une ou plusieurs files?

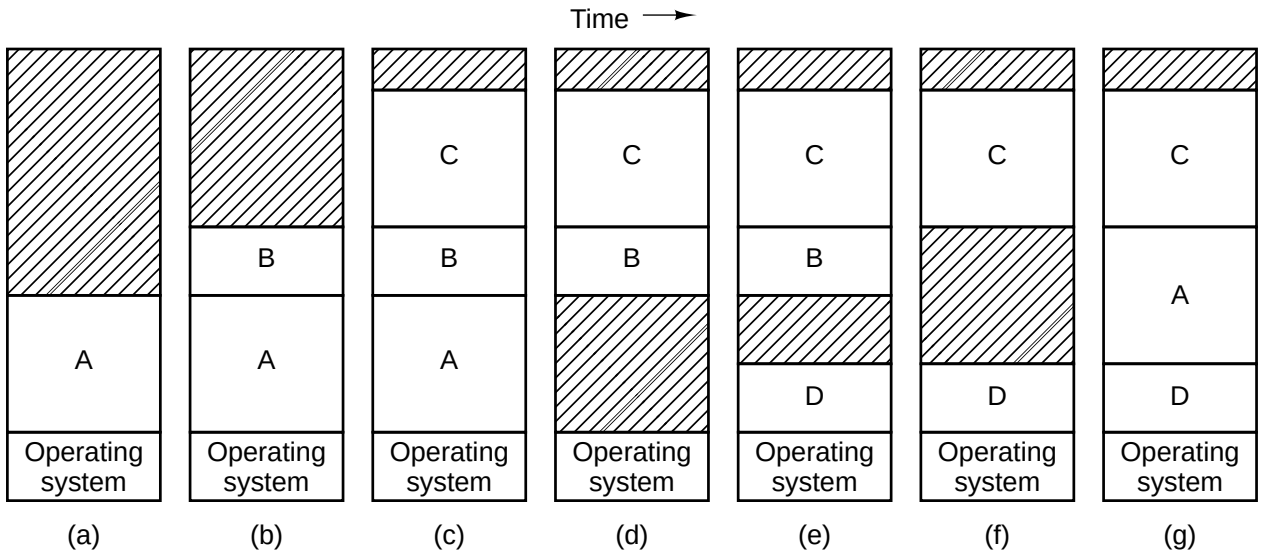
- **Inconvénient des différentes files**: la file pour une grande partition est vide tandis que celle pour une petite partition est pleine. Avec une seule file, cela peut être résolu.
- Comme il n'est pas judicieux d'attribuer une grande partition à un petit travail, autres stratégies existent:
  - Parcourir la file d'attente pour chercher le plus gros travail qui peut être placé dans la partition. Cet algorithme pénalise les petits travaux.

    Solutions:
    - Conserver au moins une **petite partition** afin que les petits travaux puissent s'exécuter sans qu'une grande partition leur soit affecté.
    - Un travail éligible **ne peut pas être ignoré plus de k fois**.
  - Les SE dont les partitions de taille fixe sont mises en place manuellement et restent inchangées par la suite a été utilisé par le OS/360 de l' IBM (*mainframes*).

Swapping

- Avec les système de temps partagé, parfois **la mémoire principale est insuffisante pour maintenir tous les processus** courant actifs: **il faut conserver les processus supplémentaires sur un disque**.
- Transfert temporaire d'un processus de la mémoire principale à une mémoire auxiliaire - **il sera ensuite ramené en mémoire pour continuer son exécution.**
- **Mémoire auxiliaire (*backing store*)**: disque rapide suffisamment grand pour pouvoir ranger les copies de toutes les images mémoire de tous les utilisateurs.
- La plupart du temps du swapping est constitué de **temps de transfert**. Le temps total de transfert est directement proportionnel à la quantité de mémoire transférée.
- Contraintes:
  - Processus à transférer doit être inactif.
  - Si les E/S accèdent d'une manière asynchrone à la mémoire utilisateur pour utiliser les buffers E/S, le processus ne peut pas être transféré.

Exemple de *swapping*



Partitions variables

- Différence principale par rapport aux partitions fixes: **leur nombre, leur localisation et leur taille varient dynamiquement au gré des allers retour des processus**.
- La **flexibilité ne dépend pas d'un nombre fixe de partitions** qui pourra être trop grand ou trop petit suivant l'utilisation de la mémoire.
- La libération et allocation de mémoire est plus compliquée.

Fragmentation

- **Fragmentation externe:** il existe un espace mémoire total suffisant pour satisfaire une requête, mais il n'est pas contigu.
- **Fragmentation interne:** la mémoire allouée peut être légèrement plus grande que le mémoire requise. Cette différence est interne à une partition mais n'est pas utilisée.
  Exemple: Trou: 18464 octets; Processus: 18462 octets
  Différence: 2 octets

Compactage: une solution pour le problème de la fragmentation externe

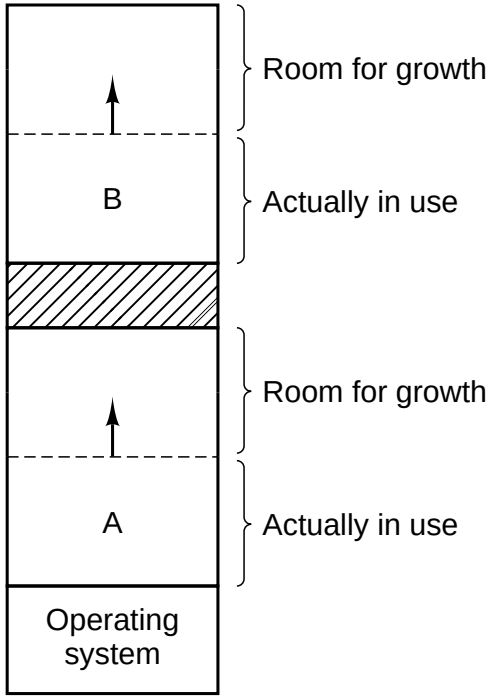
- *Objectif:* Brasser les contenus de la mémoire afin de placer toute la mémoire libre ensemble dans un seule bloc.
- Le compactage **n'est pas toujours possible**. Quand il est possible, nous devons estimer son coût.
- **L'algorithme de compactage le plus simple**: déplacer tous les processus vers une extrémité de la mémoire; tous les trous se déplacent dans l'autre direction, produisant ainsi un grand trou de mémoire disponible. **Peut être très cher!**
- La sélection d'une stratégie de compactage optimale est très difficile.

Allocation de mémoire pour un processus

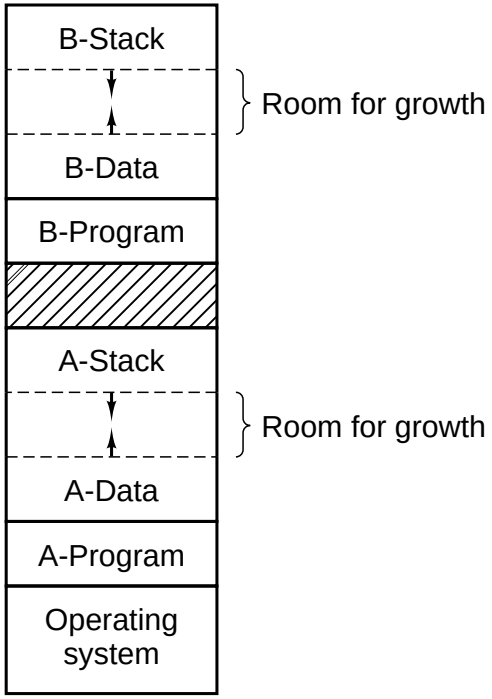
- Si le processus sont créés avec une **taille fixe**, le SE alloue **exactement la mémoire nécessaire**.
- Si les **segments de données des processus doivent croître**, par exemple par **allocation dynamique**, un problème surgit toutes les fois qu'un programme essaye de s'accroître:
  - Si **un trou est adjacent au processus** ce trou peu lui être alloué.
  - Si **le processus est adjacent à un autre processus**, le processus croissant devrait être déplacé dans un trou suffisamment grand pour lui.
  - Si **le ne peut pas croître en mémoire et que la zone de *swapping* est pleine**, le processus devra attendre ou être tué.
- S'il est à prévoir que la plupart des processus s'agrandiront lors de leur exécution, **il est bon d'allouer un peu de mémoire supplémentaire chaque fois qu'un processus est chargé ou déplacé**.
- Lorsque l'on transfère des processus sur le disque, seule la mémoire véritablement utilisée doit être recopiée.

Allocation d'espace pour l'accroissement

- Allocation d'espace pour l'accroissement d'un segment de données.
- Allocation d'espace pour l'accroissement de la pile et d'un segment de données.



(a)



(b)

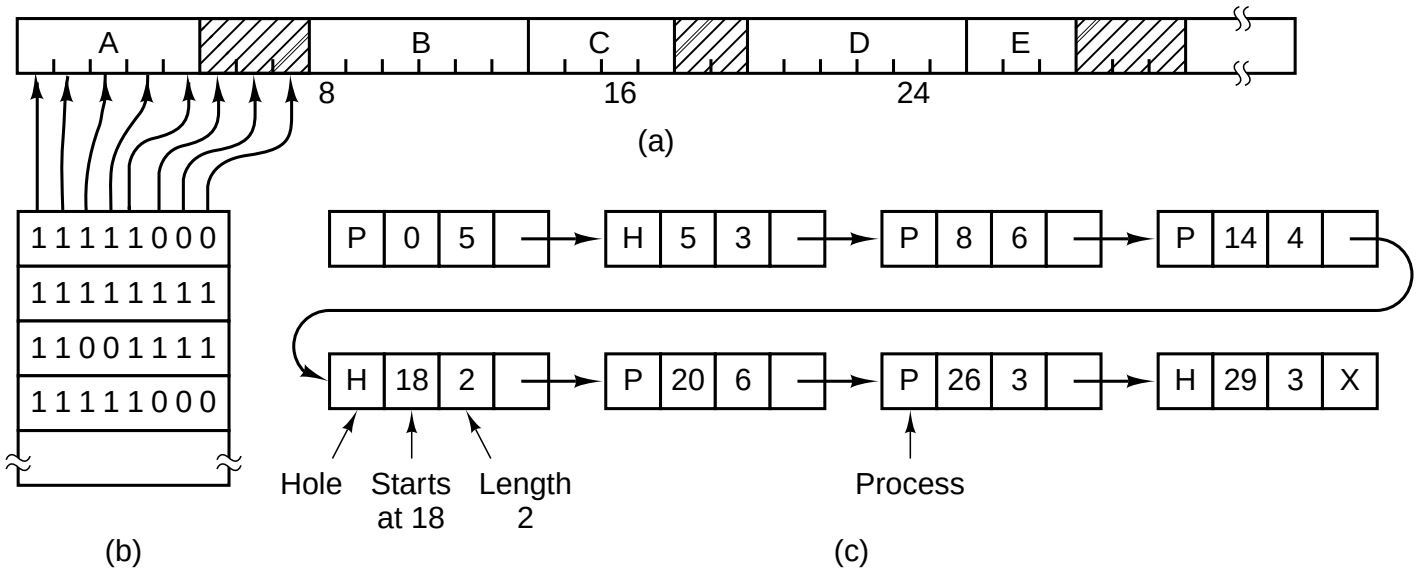
Gérer la mémoire avec des *bit maps*

- Quand la mémoire est attribué dynamiquement, c'est la SE qui doit la gérer.
- Deux manières de conserver une trace de l'utilisation de la mémoire
  1. Tables de bits (*bit maps*)
  2. Listes

Bit maps

- La mémoire est répartie en unités d'allocation dont la taille peut varier de quelques mots à plusieurs Kbytes.
- Chaque **unité d'allocation** correspond à un bit du tableau de bits, lequel est 0 **si l'unité correspondante est vide et** 1 **si elle est occupée** (ou vice-versa).

Bit maps


A
B
C
D
E
8
16
24
(a)
1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 1
1 1 0 0 1 1 1 1
1 1 1 1 1 0 0 0
(b)
P 0 5
H 5 3
P 8 6
P 14 4
H 18 2
P 20 6
P 26 3
H 29 3 X
Hole
Starts at 18
Length 2
Process
(c)

Bit maps - Taille de l'unité d'allocation

Élément fondamental dans la configuration:

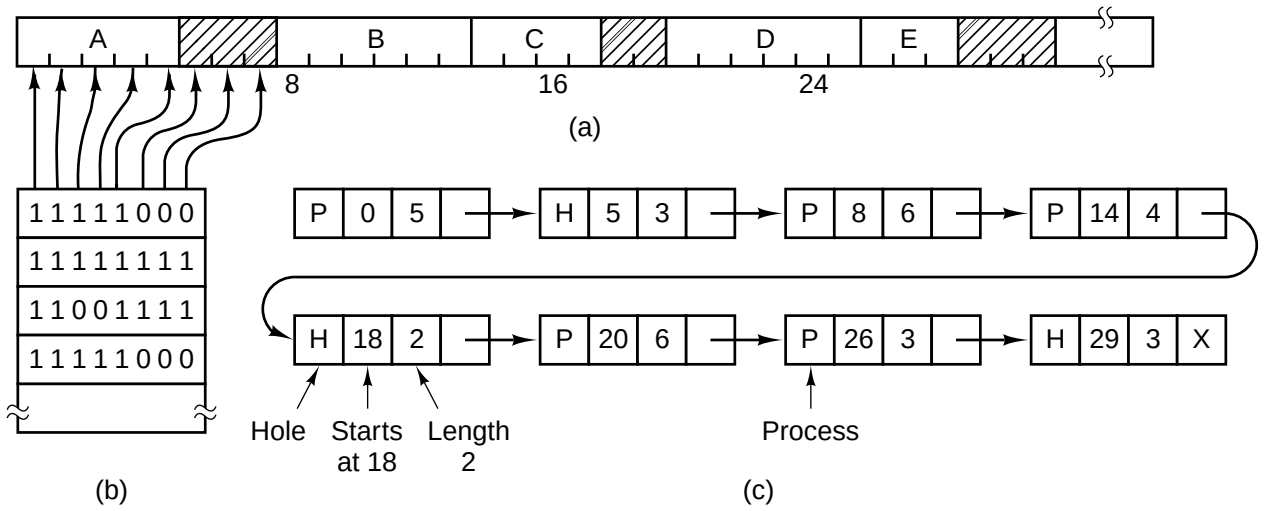
- **Plus l'unité est petite, plus le tableau de bits est important**
- Lorsque **l'unité d'allocation est plus grande, le tableau de bits est plus petit**, mais une quantité **non négligeable** de mémoire peut être gaspillée dans la dernière unité allouée à un processus ayant une taille qui n'est pas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

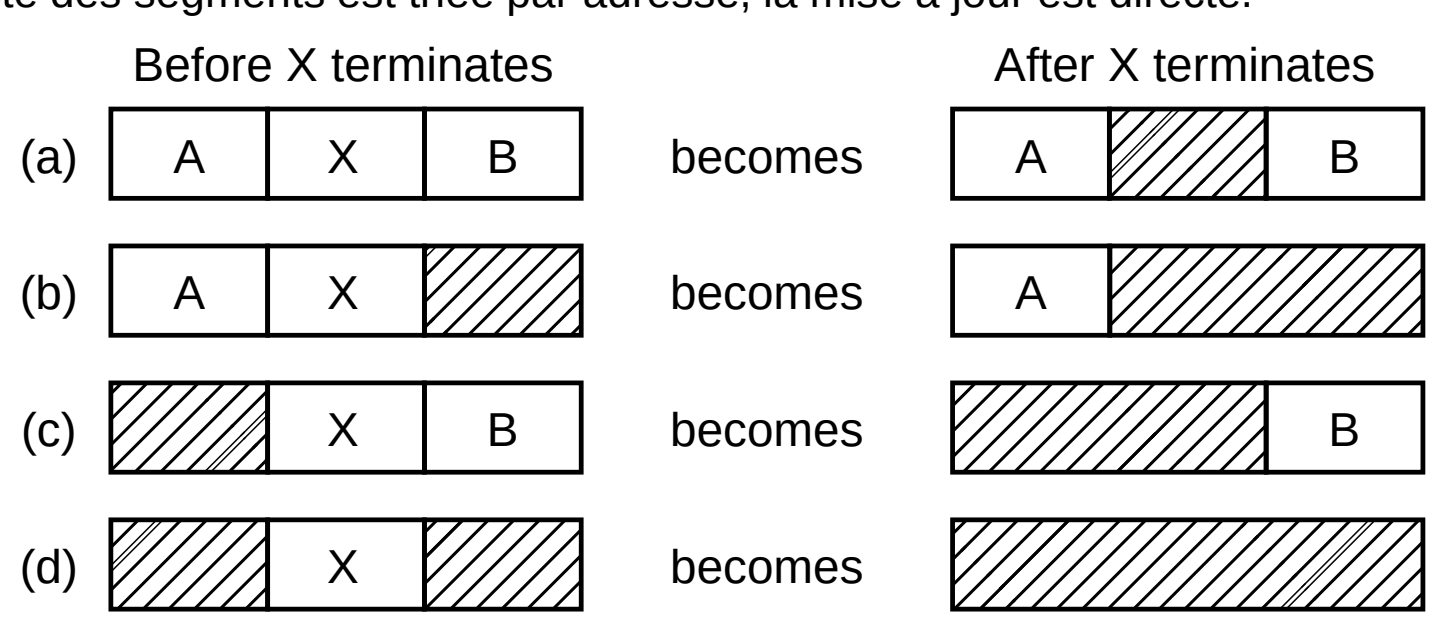
Gérer la mémoire avec des listes chaînées

- 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.



Listes chaînées

Si la liste des segments est triée par adresse, la mise à jour est directe.



Algorithmes pour allocation de mémoire - liste chaînée

Quand es processus et les trous sont indiqués dans **une liste triée par adresse**, plusieurs algorithmes peuvent être utilisés:

1. ***First-fit:*** Alloue le premier trou suffisamment grand. Le trou est ensuite divisé en deux parties, l'une destinée au processus et l'autre à la mémoire non utilisée, sauf si le processus et le trou ont la même taille.
2. ***Next-fit:*** Variation de *First-fit*. Débute la recherche de espace libre dans la liste à partir de l'endroit où il s'est arrêté la fois précédente.
3. ***Best-fit***: Alloue le trou le plus petit suffisamment grand. Plutôt que de casser un trou, il essaye de trouver un tour qui correspond à la taille demandée.
   Il est nécessaire de parcourir la liste entière à moins que la liste ne soit triée par taille.
   Cette stratégie fait rester le trou le plus petit.
4. ***Worst-fit***: Alloue le trou le plus grand (parcours de toute la liste).

Comparaison des algorithmes

- *Best-fit* est plus lent que *First-fit* et il perd plus de place mémoire que *First-fit* et *Next-fit*: **il tend à remplir la mémoire avec des minuscules trous inutiles.**
- La rapidité de algorithmes peut être améliorée si l'on établit des listes séparées pour les processus et les trous.
  Contrepartie: complexité plus grande et ralentissement quand la mémoire est libérée: un segment qui est libéré doit être effacé de la liste des processus et inséré dans la liste des trous.
- Liste des trous doit être triée par taille afin d'augmenter la rapidité de l'algorithme du *Best-fit*.
- Dans le cas d'une liste de trous triée par taille, le *Best-fit* et le *First-fit* sont aussi rapides l'un que l'autre et le *Next-fit* est inutile.

Mémoire virtuelle

- Principe de la **mémoire virtuelle**: la taille de l'ensemble formé par le programme, les données et la pile **peut dépasser la capacité disponible de mémoire physique.**
- Le SE conserve les parties de programme en cours d'utilisation dans la mémoire principale et le reste sur le disque.
- La mémoire virtuelle peut aussi fonctionner dans un système multiprogrammé.

Mémoire virtuelle

- Séparation entre la mémoire logique de l'utilisateur et la mémoire physique.
- Seulement une partie d'un programme a besoin d'être dans la mémoire pour l'exécution.
- Les programmes peuvent être plus grands que la mémoire physique.
- Rend plus facile la tâche de programmation: le programmeur n'a pas besoin de se préoccuper de la quantité de mémoire physique disponible.

Implantation

- Pagination à la demande
- Segmentation à la demande

Pagination (*Paging*)

- L'espace adresse logique d'un processus n'est pas contigu.
- La mémoire physique est découpée en blocs de taille fixe appelés **cadres de pages (*frames*)**.
  Taille: puissance de 2 $\Rightarrow$ entre 512 bytes et 8192 bytes.
- La mémoire logique est également subdivisée en blocs de la même taille appelés **pages**.

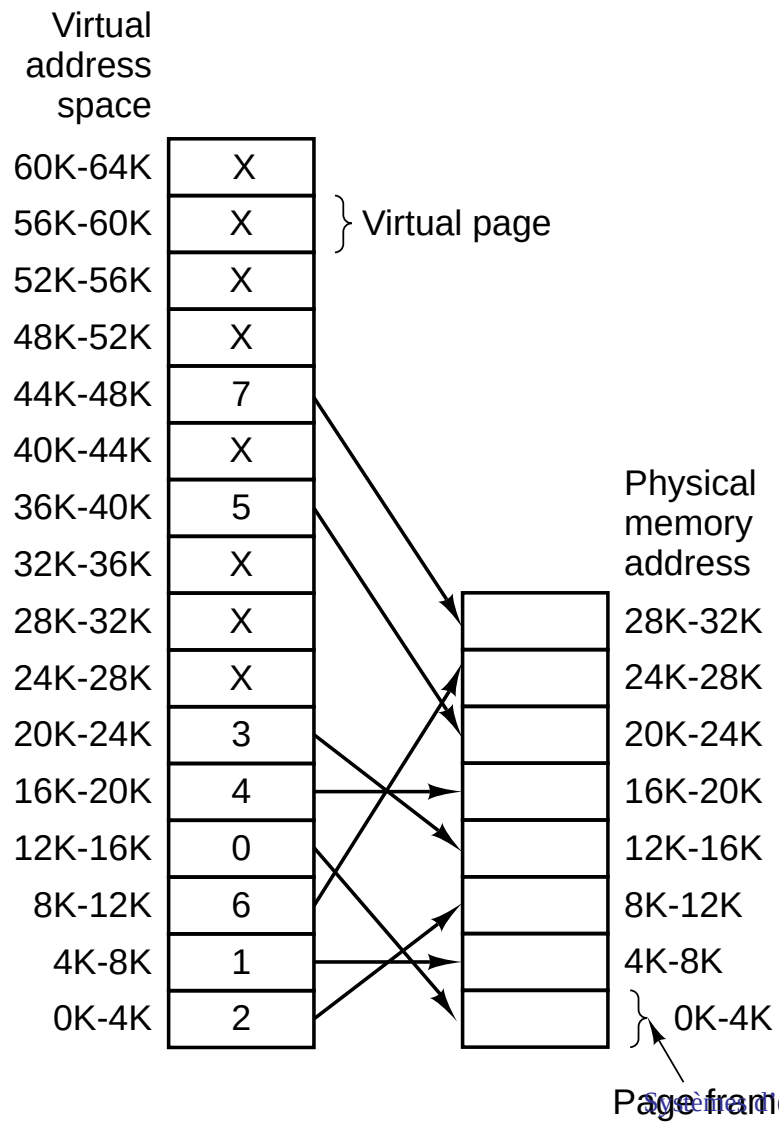
Taille pages = **Taille cadres**

- Pour exécuter un programme de n pages, il faut trouver n cadres libres et charger le programme.
- Il faut savoir quels cadres sont alloués, lesquels sont disponibles, etc.
  Cette information est dans la **table de cadre (*Frame table*)**
- **Table de pages:** traduit l'adresse logique en adresse physique.

Exemple

- Adresses virtuelles: Un ordinateur peut produire des adresses sur 16 bits avec des valeurs entre 0 et 64 KBytes.
- L'ordinateur a seulement 32 KBytes de mémoire physique.
- La taille de page et de cadre est de 4 KBytes.
- Nous avons donc 16 pages virtuelles et 8 cadres de pages.

Exemple


Virtual address space
60K-64K
56K-60K
52K-56K
48K-52K
44K-48K
40K-44K
36K-40K
32K-36K
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
X
X
X
X
7
X
5
X
X
X
3
4
0
6
1
2
Virtual page
Physical memory address
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
Page frame

Pagination à la demande

- Processus réside dans la mémoire secondaire (disque).
  1. Nous voulons l'exécuter.
  2. *Swap in*: processus dans la mémoire primaire.
  3. Il n'est pas nécessaire de faire *swap* du processus dans sa totalité (*lazy swapper*).

  Transfert d' une page en mémoire seulement quand il est nécessaire
- **Support matériel:** possibilité de distinguer entre les pages qui sont en mémoire et celles qui sont sur disque.
  - **Bit présence/absence**
    1 : page en mémoire
    0: sur disque
- L'accès à une page marquée invalide provoque un déroutement de **défaut de page** (*page fault*)

Défaut de page

1. S'il y a une référence à une page marquée absente alors déroutement *page fault* (SE)
2. Trouver un cadre de page libre.
3. "Swap" la page dans le cadre.
4. Modifier la table interne et la table de pages afin d'indiquer que la page est maintenant en mémoire.
5. Redémarrer l'instruction interrompue par le déroutement d'adresse illégale. Le processus peut accéder à la page comme si elle avait toujours été en mémoire.

Exemple

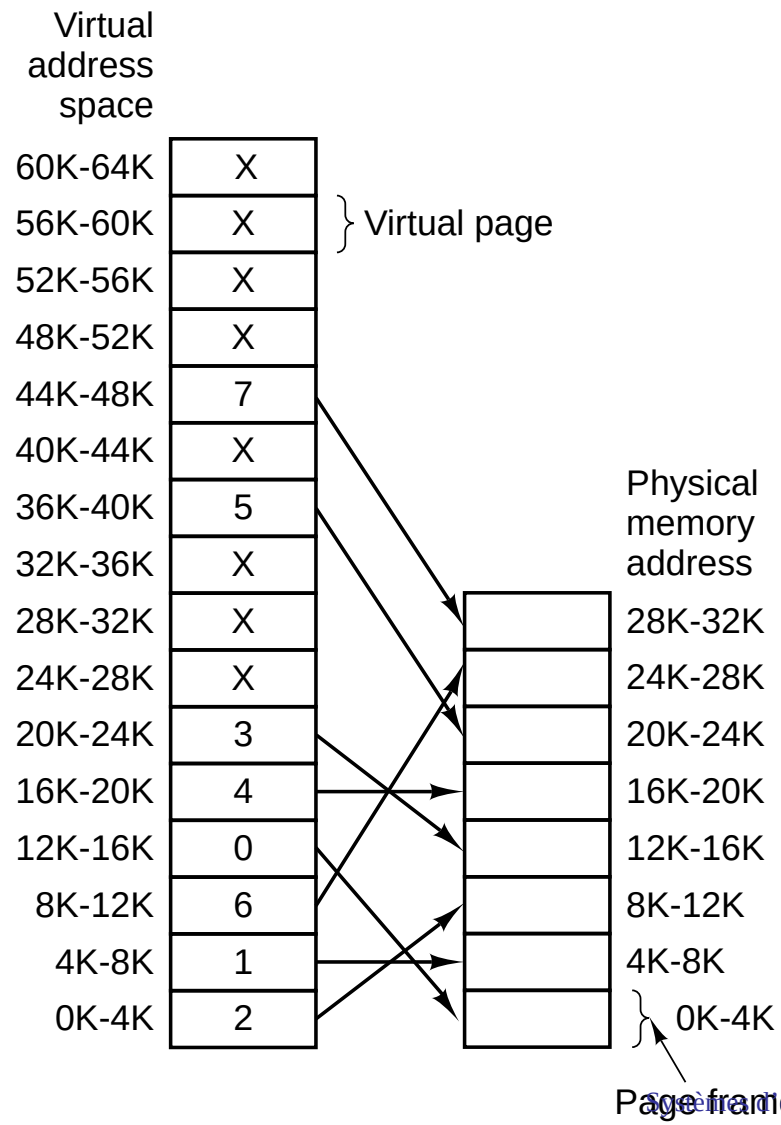
Que se passe-t-il si le programme essaye par exemple de faire appel à une page non présente, avec l'instruction

```
MOV REG, 32780
```

qui correspond à l'octet 12 de la page virtuelle 8 (qui commence à 32768)?

- La MMU remarque que la page est absente et fait une déroutement de page.
- SE décide de déplacer le cadre de page 1, il doit charger la page virtuelle 8 à l'adresse physique 4K et faire deux changements de correspondance dans la MMU.
  1. Marquer l'entrée de la page virtuelle 1 comme non utilisée.
  2. Changer le bit présence/absence de la page 8.
  3. Quand l'instruction déroutée sera ré-exécutée, il mettra l'adresse virtuelle 32780 en correspondance avec l'adresse 4108.

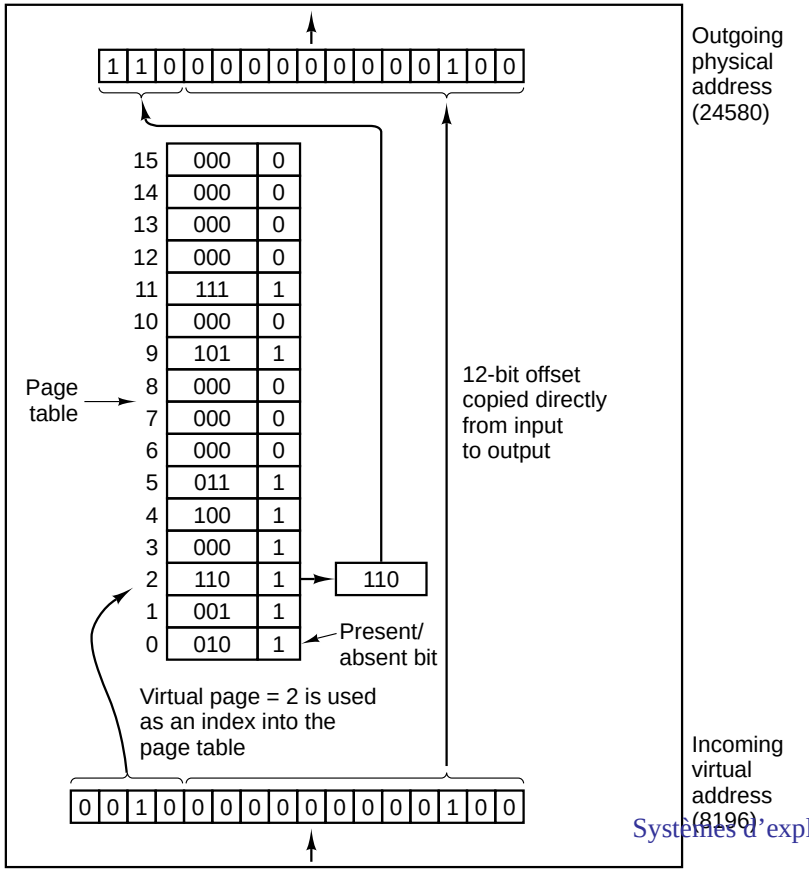
Exemple


Virtual address space
60K-64K
56K-60K
52K-56K
48K-52K
44K-48K
40K-44K
36K-40K
32K-36K
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
X
X
X
X
7
X
5
X
X
X
3
4
0
6
1
2
Virtual page
Physical memory address
28K-32K
24K-28K
20K-24K
16K-20K
12K-16K
8K-12K
4K-8K
0K-4K
Page frame

Exemple: adresse virtuelle

L'adresse virtuelle de 16 bits est divisée en deux parties:

- un numéro de page sur 4 bits (donc nous pouvons avoir 16 pages)
- un décalage (*offset*) sur 12 bits



Adresse virtuelle

- Le numéro de page est utilisé comme un index dans la **table de pages**
- Si le bit de présence/absence est à 0 un déroutement vers le SE est mis en place.
- Si le bit de présence/absence est à 1, le numéro de cadre de page trouvé dans la table de pages est copié dans le registre de sortie. On y ajoute les 12 bits de décalage ou déplacement (*offset*).
- L'adresse physique est formée par ces deux parties ensembles.

Table de page

- L'adresse générée par la CPU est divisée en 2 parties:
  1. Numéro de page (*Page number*) p:
     Indice de la table de pages laquelle contient l'adresse de base pour chaque page.
  2. Déplacement dans la page (*Offset*) d:
     Combiné avec l'adresse de base définit l'adresse physique qui est envoyée à l'unité de mémoire.
- Le rôle d'une table de pages est de faire correspondre des pages virtuelles à des cadres de pages.
- **Mathématiquement, une table des pages est une fonction, qui a comme argument le numéro de la page virtuelle et comme résultat le numéro du cadre physique.**

Table de pages

Deux problèmes majeurs doivent être soulevés et représentent des contraintes importantes dans la construction des ordinateurs:

- La table des pages doit être extrêmement grande.
  - Ordinateur modernes utilisent des adresses virtuelles d'au moins 32 bits.
  - Table de pages avec environ un million d'entrée!
  - Chaque processus a besoin de sa propre table de pages.
- La correspondance doit être rapide.
  - Pour chaque instruction il est nécessaire de faire référence à la table des pages 1 fois, 2 fois et parfois plus.

Implantation de la table de pages

Manière la plus simple

- Table de pages dans la mémoire principale.
- *Page Table Base Register* (PTBR): indique la table de pages
- Commutation de contexte: chaque processus possède sa table de pages
  Pour modifier la table de pages (cad, changer la fonction de traduction) il suffit de modifier PTBR.
- Problème: temps requis pour charger la table des pages en totalité à chaque changement de contexte.

Implantation de la table de pages

Solution standard

- Utiliser les *registres associatifs* (ou *translation look aside buffers: TLB*): petite mémoire cache matérielle spéciale à la consultation rapide.
  - Contiennent seulement quelques entrées de la table
  - Recherche en parallèle

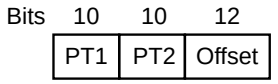
Page number	Frame number

- Traduction de l'adresse (A, A')
  Si A' est dans les registres associatifs
  alors le numéro de cadre est disponible immédiatement
  sinon obtenir le numéro de cadre à partir de la table de pages en mémoire
- Si le TLB est plein, le SE doit sélectionner une entrée devant être remplacée

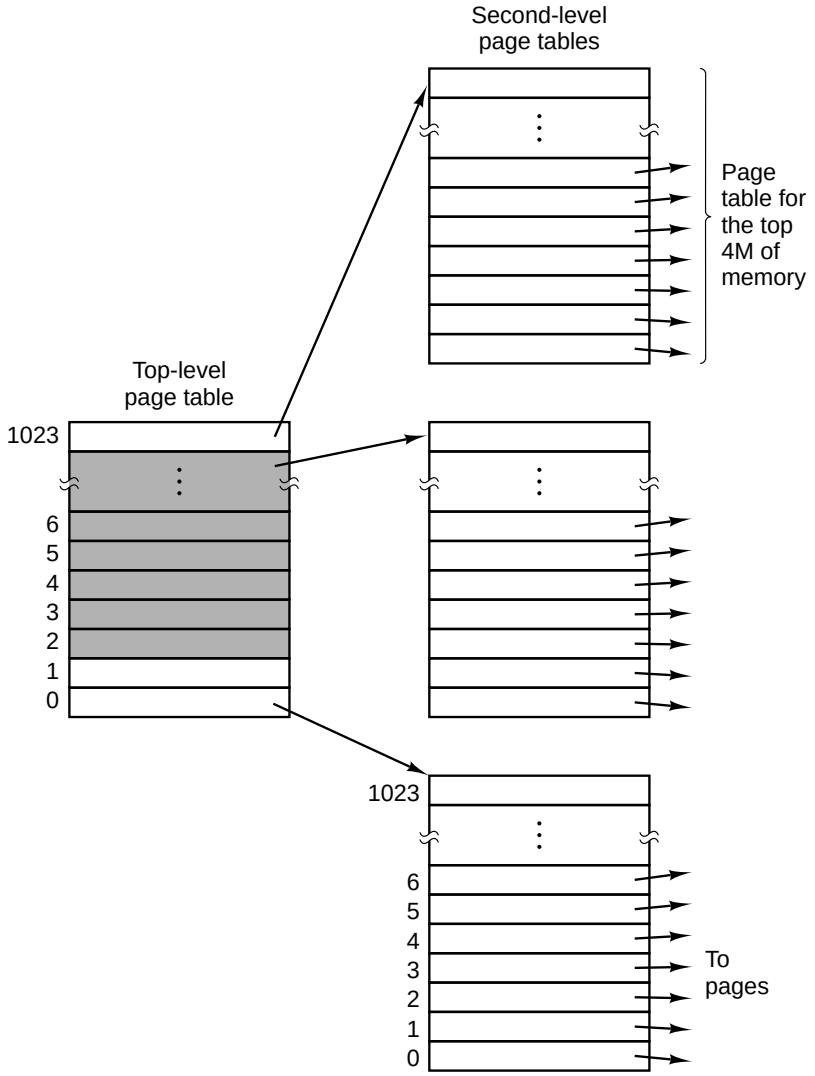
Pagination à multi niveaux *(Multilevel Paging)*

- La plupart des SE moderne supportent un espace logique très grand (table de pages excessivement grande)
- Solution: **Subdiviser les pages**
  1. Pagination à 2 niveaux
  2. Généralisation: plusieurs niveaux
- Performance: Chaque niveau est stocké comme une table séparée en mémoire. Ainsi, la conversion d'une adresse logique à une adresse physique peut demander 4 accès mémoire.
- Mémoire cache permet une performance raisonnable.

Exemple


Bits 10 10 12
PT1
PT2
Offset

(a)

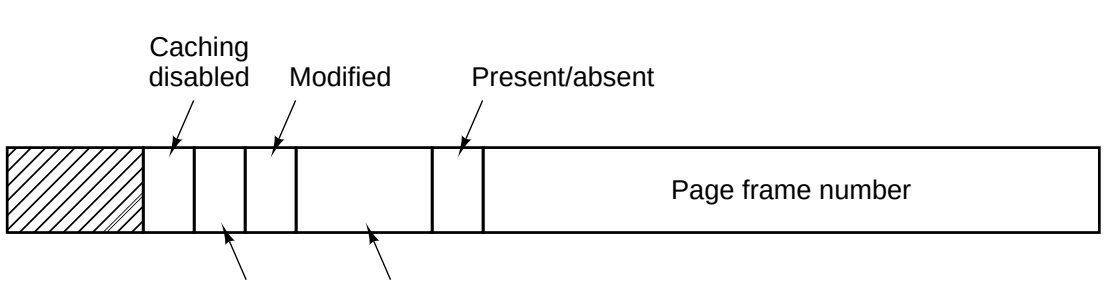

Second-level
page tables
Page
table for
the top
4M of
memory
Top-level
page table
1023
6
5
4
3
2
1
0
1023
6
5
4
3
2
1
0
To
pages

(b)

Exemple

- Adresse virtuelle de 32 bits partitionnée en deux champs de 10 bits ($PT1$ et $PT2$) et un champs de 12 bits (*offset*).
- *offset* de 12 bits, donc pages de $4KBytes$
  Total de 2^{20} pages
- Table de pages niveau 1, avec 1024 entrées, correspondant au champs $PT1$.
- L'entrée localisée par l'indexation de la table des pages de niveau 1 produit l'adresse ou le numéro de cadre de page de la table de pages du niveau 2.
- L'entrée 0 de la table de page du niveau 1 pointe sur la table des pages du corps du programme.
- L'entrée 1 de la table de page du niveau 1 pointe sur la table des pages des données.
- L'entrée 1023 de la table de page du niveau 1 pointe sur la table des pages de la pile.
- Les autres (grisés) ne sont pas utilisées.
- Le champs $PT2$ est utilisé comme index dans la table de page du niveau 2 sélectionné pour trouver le numéro de cadre de page de la page elle-même.

Structure d'une entrée de table des pages



- La taille varie d'une machine à l'autre.
- Champ le plus important est le numéro de cadre de page (le but est de le trouver).
- Bit de **présence/absence**: quand il est à 0 la page virtuelle à laquelle l'entrée appartient n'est pas en mémoire.
- Bits de **protection** précisent quelle sorte d'accès sont permis. Dans sa forme la plus simple ce champ contient un seul bit (valeur 0 pour lecture/écriture et 1 pour lecture seulement). Un schéma plus sophistiqué propose 3 bits (un pour la lecture, un pour l'écriture et un pour l'exécution).

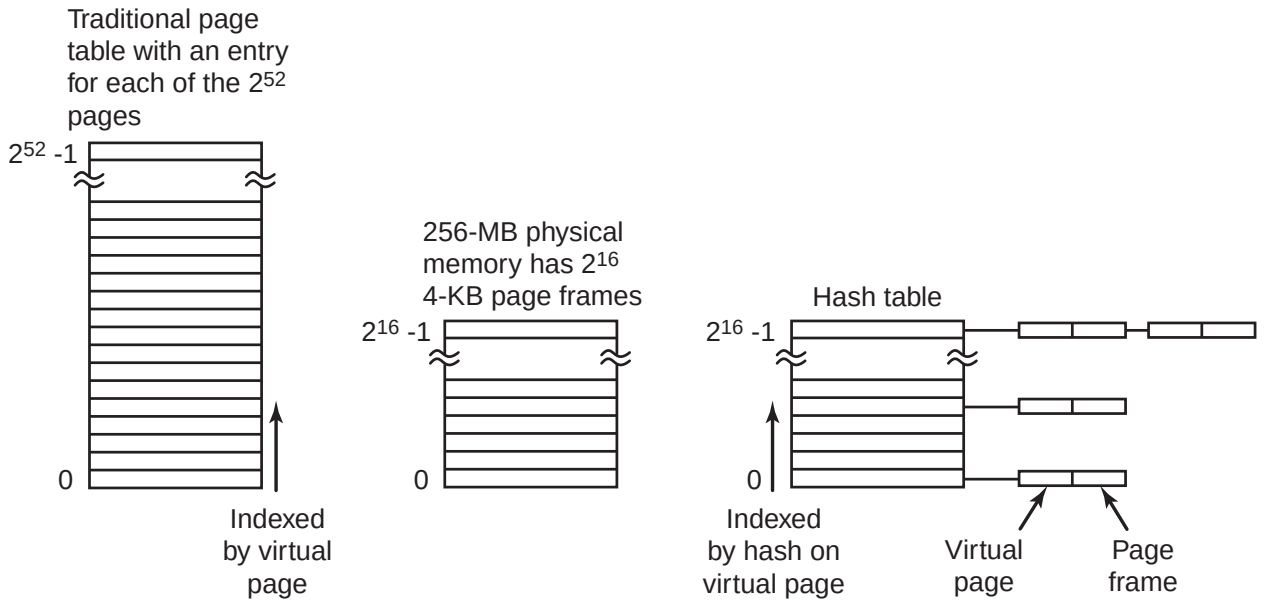
Structure d'une entrée de table des pages

- Bits **modifiés et référencés** conservent une trace de l'utilisation de la page.
- **Dirty bit** (modifié): Quand une page est accédée en écriture, le bit est mis à 1. Ce bit est nécessaire quand le SE décide de récupérer un cadre de page:
  - Si la page a été modifiée, elle doit être écrite dans le disque.
  - Si elle n'est pas modifiée, on peut juste l'abandonner.
- **Bit référencé**: Mis à 1 chaque fois qu'un page est consultée, que se soit en lecture ou en écriture. Cette valeur aide le SE à choisir la page à évincer quand un défaut de page est généré.
  Les pages qui ne sont pas en cours d'utilisation sont des meilleurs candidates que les autres et ce bit joue un rôle important dans plusieurs algorithmes de remplacements de page.
- **Bit qui permet d'inhiber le cache** pour une page.

Table de pages inversée

- Une entrée pour chaque page réelle (cadre).
- Chaque entrée est constituée de l'adresse virtuelle de la page stockée dans cet emplacement de mémoire réelle, avec de l'information à propos du processus auquel appartient cette page.
- Une seule table de pages dans le système: elle possède une seule entrée pour chaque page de mémoire physique.
- Réduction de la quantité de mémoire requise pour stocker chaque table.
- Augmentation de la quantité de temps nécessaire pour chercher la table quand il se produit une référence à une page.
- Utilisation d'une table de hashage (*hash table*) pour limiter la recherche à une - ou à peu - d'entrées dans la table.

Table de pages traditionnelle × table de pages inversée



Remplacement de pages

Quand une page n'est pas en mémoire...

1. Trouver l'**emplacement de la page désirée sur disque**.
2. Trouver un **cadre de page libre**.
   (a) S'il existe un cadre libre, l'utiliser.
   (b) Sinon, utiliser un **algorithme de remplacement de pages** pour sélectionner un cadre (victime).
   (c) Enregistrer la page victime dans le disque, modifier les tables de pages et de cadres.
3. **Lire la page désirée dans le cadre libéré**, modifier les tables de pages et de cadres.
4. Redémarrer le processus utilisateur.

Algorithmes de remplacement des pages

- Nous souhaitons l'algorithme avec le taux de défaut de page le plus bas.
- Évaluation: exécuter l'algorithme sur une séquence particulière (*reference string*) de références mémoire et calculer le nombre de défauts de pages.

Algorithme optimal

- Remplacer la page qui mettra le plus de temps à être de nouveau utilisée.
- **Impossible à implanter**: il requiert une connaissance future de la chaîne de références.
- Utilisé pour effectuer des études comparatives.

Algorithme FIFO

- Associe à chaque page le moment où l'on a ramené cette page en mémoire.
- Quand on doit remplacer une page, c'est la plus ancienne que l'on sélectionne.
- Sa performance n'est pas toujours bonne.
- *Belady's anomaly*
  plus de cadre $\not\Rightarrow$ moins de défaut de pages

Algorithme LRU (*least-recently used*)

- Remplacer la page que n'a pas été utilisée pendant la plus longue période de temps.
- Très utilisé.
- **Problème**: savoir comment implanter, demande une forte assistance matérielle.
- Implantation via compteurs:
  - Chaque page possède un compteur.
  - Chaque fois que l'on effectue une référence à une page les contenus de l'horloge sont copiés dans le compteur.
  - Nous remplaçons la page avec le moment d'utilisation le plus petit.
- Implantation par pile:
  - Maintenir une pile de numéro de pages.
  - Chaque fois que l'on référence une page, on la supprime et on la met au somment de la pile (sommet $\Rightarrow$ la pile récemment utilisée; base $\Rightarrow$ page LRU)
  - Implantation comme une liste à double chaînage avec un pointeur sur la tête et un autre sur la queue.

Algorithme de comptage

- Compteur avec le nombre de références à chaque page
- LFU (*least frequently used*)
- MFU (*most frequently used*)

Allocation de cadres de pages

- Nombre de cadres de pages (*frames*): défini par l'architecture.
- Nombre maximum: déterminé par la quantité de mémoire physique disponible.

Allocation équitable

- m cadres
- n processus
- m/n cadres pour chaque processus

Allocation de cadres de pages

Allocation proportionnelle

- Allouer la mémoire disponible à chaque processus selon sa taille.
- s_i: taille de la mémoire virtuelle pour p_i.
- $S = \sum s_i$.
- m: nombre total de cadres.
- a_i: cadre alloués à p_i.
- $a_i = s_i / S * m$.
- Nous devons arrondir chaque a_i pour que ce soit un entier supérieur au nombre minimal de cadres requis par le jeu d'instructions. La somme des a_i ne doit pas dépasser m.

Allocation prioritaire

Utiliser un schéma d'allocation proportionnelle où la quantité de cadres dépend non de la taille des processus mais de leurs priorités.

Allocation globale et allocation locale

- **Remplacement global:** un processus sélectionne un cadre de page de remplacement à partir de l'ensemble de **tous** les cadres de pages
  - Un processus ne peut pas contrôler son propre taux de défaut de pages
  - L'ensemble de pages en mémoire pour un processus ne dépend pas seulement du comportement de la pagination pour ce processus, mais aussi du comportement de la pagination pour les autres processus
- **Remplacement local:** un processus sélectionne un cadre de page de remplacement à partir de **son propre ensemble** des cadres de pages
  - Le nombre de cadres alloués à un processus ne change pas

$\Rightarrow$ Méthode courante: Remplacement global

Exemple: Allocation globale et allocation locale

	Age
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(a)

A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

(b)

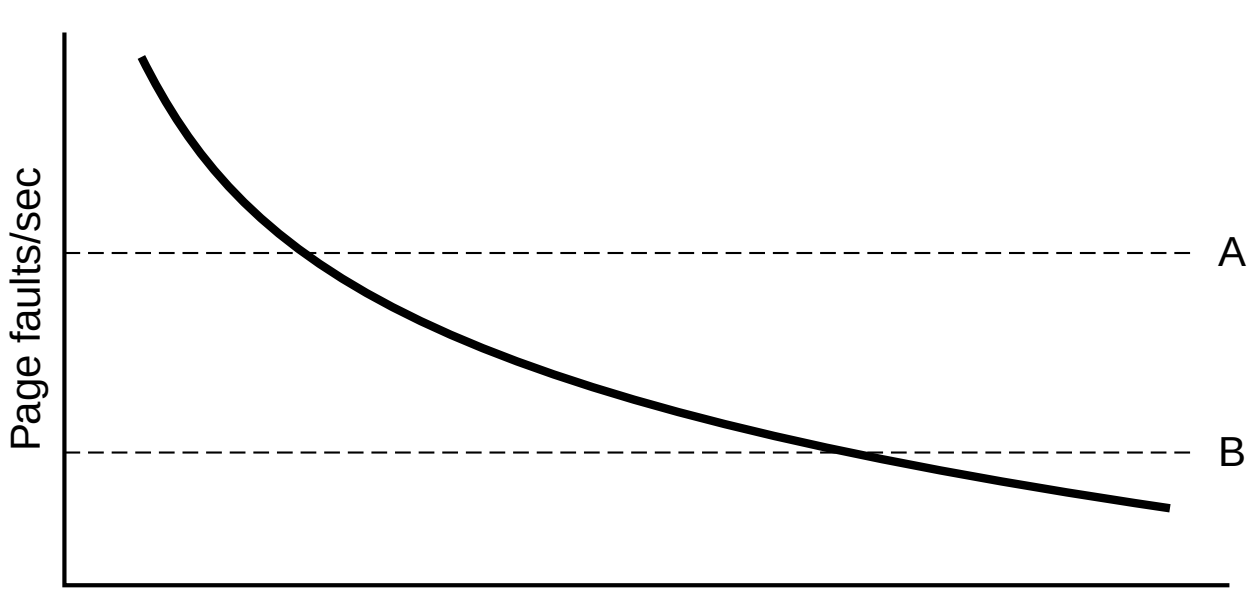
A0
A1
A2
A3
A4
A5
B0
B1
B2
A6
B4
B5
B6
C1
C2
C3

(c)

Page Fault Frequency

- Si un algorithme global est employé, il est possible de **démarrer chaque processus avec un nombre de pages proportionnel à la taille du processus**, mais **l'allocation devra être réajustée dynamiquement pendant l'exécution du processus**.
- Pour gérer cette allocation: algorithme **Page Fault Frequency**
- Le PFF indique quand augmenter ou dimunuer l'allocation de page pour un processus. Contrôle ta **taille de l'ensemble de l'allocation**.
- Pour la plupart des algorithmes de remplacements de page le nombre de défauts de pages diminue lorsque le nombre de pages allouées augmente.
- Pour mesurer le taux de défauts de page: compter le nombre de défauts de pages par seconde.
- Limite supérieure: taux excessivement élevé de défauts de pages.
- Limite inférieure: taux de défauts de pages si base que nous pouvons déduire que le processus a trop de mémoire.
- PFF essaie de maintenir un taux de pagination dans les limites acceptables.

Page Fault Frequency



Écroulement (*Trashing*)

- Haute activité de pagination: un processus passe plus de temps à paginer qu'à exécuter.
- Problème de performance:
  - Diminution de l'utilisation de la CPU.
  - SE pense qu'il doit augmenter le niveau de multiprogrammation.
  - Un nouveau processus est ajouté au système.
- Pour prévenir l'écroulement, nous devons fournir à un processus autant de cadres qu'il en aura besoin.
- Plusieurs moyens pour savoir combien de cadres un processus a besoin.
  - La stratégie de l'ensemble de travail commence par la définition du modèle de localisation de l'exécution d'un processus.
  - **Modèle de localisation:** Lorsqu'un processus s'exécute, il se déplace de localisation en localisation
  - Localisation: Ensemble de pages utilisées activement ensemble
  - Un programme est généralement constitué de plusieurs localisations qui peuvent se chevaucher.

Modèle de l'ensemble de travail

- Le modèle de l'ensemble de travail est fondé sur l'hypothèse de la localisation.
- **Paramètre Δ:** fenêtre de l'ensemble de travail (**working-set window**)
  $\Rightarrow$ Il s'agit d'examiner les Δ références aux pages les plus récentes.
- **Ensemble de travail:** ensemble de pages dans les Δ références aux pages les plus récentes.
- L'exactitude de l'ensemble de travail dépend de Δ:
  1. Δ trop petit
     il ne renfermera pas toute la localisation
  2. Δ trop grand
     il peut chevaucher plusieurs localisations
  3. $\Delta = \infty$
     l'ensemble de travail est la totalité de pages référencées pendant l'exécution du programme

- wss_i (*Working-set size of process* P_i): taille de l'ensemble de travail (nombre de pages).
- La demande totale de cadres est

$$D = \sum(wss_i)$$

- Si $D > m$ alors *Trashing* (m : nombre de cadres disponibles).
- L'emploi du modèle *Working-set*
  - SE surveille le *Working-set* de chaque processus et alloue à cet *Working-set* les cadre suffisants (correspondant à wss_i).
  - Si $D > m$ alors le SE sélectionne un processus à arrêter.
- Cette stratégie de l'ensemble de travail évite l'écroulement tout en gardant le degré de multiprogrammation le plus haut possible.
- Difficulté: surveillance

Autres considérations

Le choix d'un algorithme de remplacement et la politique d'allocation sont les décisions principales à effectuer pour un système de pagination. Il existe également beaucoup d'autres considérations

- Pré-pagination
- Taille de la page
- Structure des programmes
- Verrouillage des E/S

Segmentation

- L'utilisateur préfère voir la mémoire comme un ensemble de segments de taille variable, sans qu'il ait nécessairement un ordre entre les segments.
- La segmentation est un schéma de gestion de la mémoire qui supporte cette vision.
- Un espace adresse logique est un ensemble de segments.
- Chaque segment possède un nom (les segments sont numérotés) et une longueur.
- Adresse logique:
  ⟨ numéro de segment, déplacement⟩

Segmentation

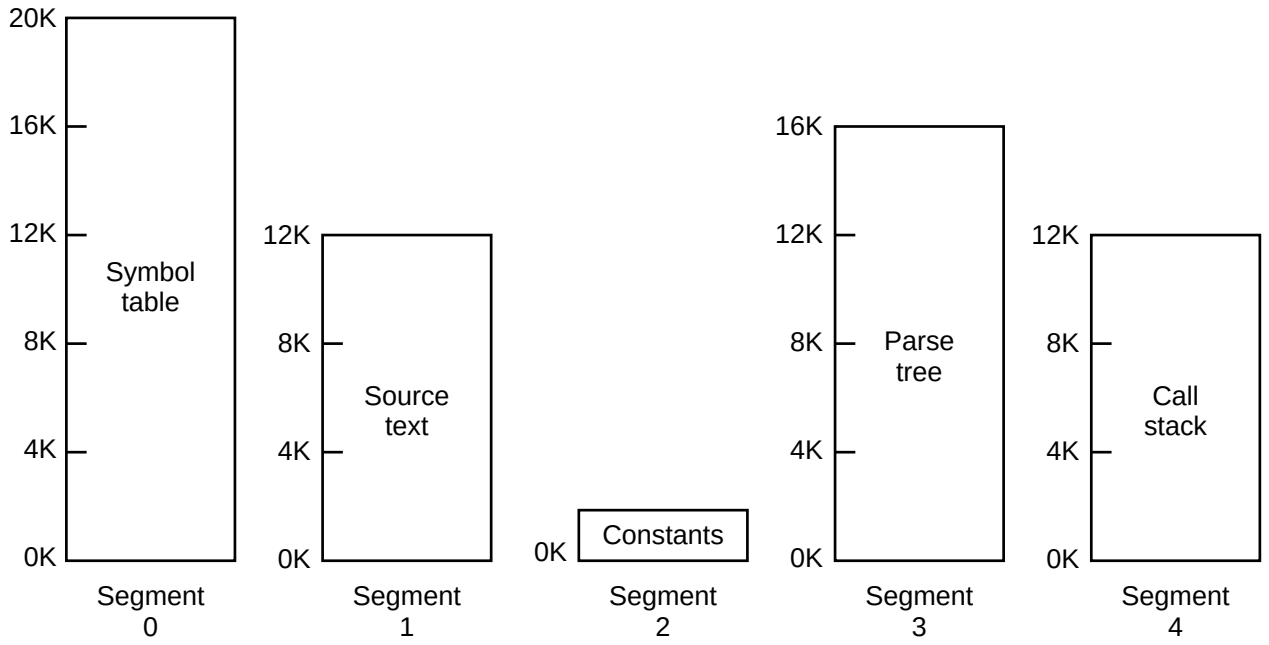
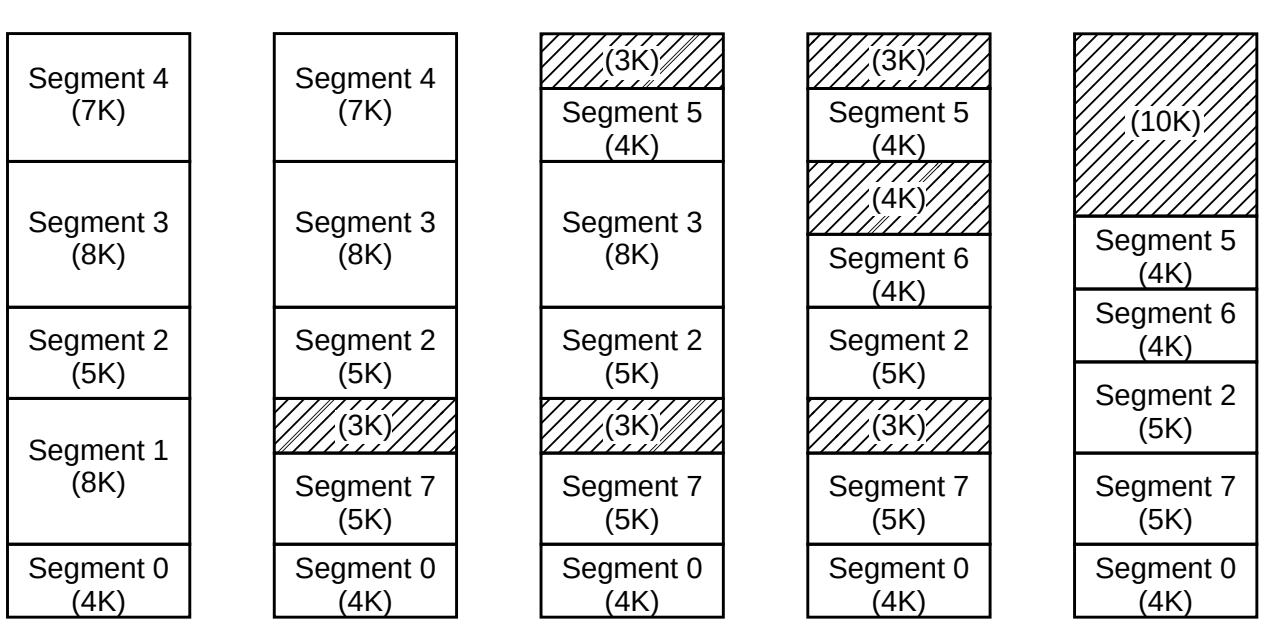


Table de segments (*Segment Table*)

- Transforme des adresses à deux dimensions (définies par l'utilisateur) en adresse à une dimension.
- Chaque adresse possède:
  1. **Base:** Contient l'adresse physique de début où le segment réside en mémoire
  2. **Limite:** spécifie la longueur du segment

Implémentation: segmentation pure



Implémentation

- *Segment table base register (STBR):* pointe vers la table de segments
- *Segment table length register (STLR):* indique la longueur de la table de segments
- Adresse logique (s, d):
  s est légale si $s < STLR$
  Si s est légale alors calculer l'adresse physique en utilisant $STBR$
- Ralentissement $\Rightarrow$ solution: registre associatifs

Protection et partage

Protection

- Association de la protection avec le segment
- Le matériel de conversion de la mémoire vérifie les bits de protection associés à chaque entrée de la table de segments pour empêcher des accès illégaux à la mémoire

Partage

- Les segments sont partagés quand les entrées dans les tables de segments de 2 processus différents pointent vers les mêmes emplacement
- Chaque processus possède une table de segments

Segmentation × Pagination

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Segmentation avec pagination

- Microprocesseurs:
  1. Ligne Motorola: 68000
  2. Intel 80X86
- Modèle de mémoire fusionné (mélange de pagination et segmentation)
- Cette combinaison est mieux illustrée par 2 architectures différentes
  1. MULTICS
  2. OS/2 (version 32 bits)