Deuxième partie

Gestion de la mémoire

- Introduction et concepts
- Mémoire uniforme vs. mémoire hiérarchisée
- Mémoire uniforme
  - Partition unique
  - Plusieurs partitions de tailles différentes
  - pagination
  - Segmentation
  - Segmentation paginée
- Mémoire hiérarchisée (virtuelle)
  - Swapping
  - Pagination à la demande
  - Algorithmes de remplacement de pages
  - Allocation de cadres de pages
  - Fichiers mappés en mémoire
- Etude de cas
  - Le Pentium
  - Linux

Introduction

- Pour être exécutées, les instructions d'un programme doivent être chargées en mémoire centrale
- Multiprogrammation : plusieurs programmes sont chargés en même temps
  → La mémoire est une ressource importante d'un système informatique
- Problème d'allocation des ressources : la mémoire est finie et généralement trop petite pour loger tout . . .
- La partie du SE qui s'occupe de l'allocation de la mémoire s'appelle le gestionnaire de la mémoire

Objectifs

- Optimisation de l'utilisation de la mémoire principale
- Augmenter le rendement global du système
  - Exp. suffisamment de processus en mémoire pour assurer une bonne répartition entre les cycles E/S et CPU

Introduction

Fonctions d'un gestionnaire de la mémoire

- garder trace de l'état de chaque portion de la mémoire (libre ou allouée)
- une politique d'allocation
  - choisir entre plusieurs processus
  - allouer de l'espace mémoire à un processus donné
- une politique de libération

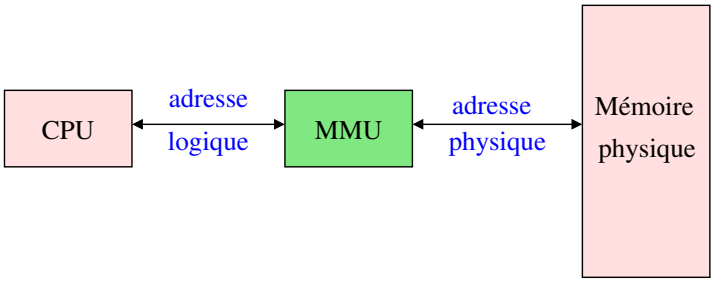
Rappel. Hiérarchie mémoire

- l'UC a un accès direct à ses registres et à la mémoire principale :
  - un cycle d'horloge (ou moins) pour accéder à un registre
  - souvent, plusieurs cycles d'horloge sont nécessaires (suivant le type d'adressage) pour accéder à la mémoire principale
- mémoire plus rapide = mémoire plus petite parce que plus coûteuse
- le cache se situe entre les registres et la mémoire principale
- l'accès au disque dur (mémoire secondaire) est plus lent par rapport à la mémoire centrale
- le matériel et le système d'exploitation sont responsables du déplacement des objets le long de cette hiérarchie :
  - exemple : de la mémoire centrale au disque et inversement.

Concepts

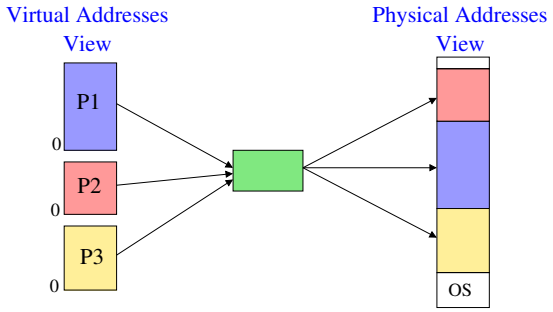
Adresses physiques vs. logiques

- adresse physique (réelle) correpond à la vision de la mémoire centrale → espace d'adressage physique ou réel
- adresse logique (virtuelle) est l'adresse générée par l'unité centrale → espace d'adressage logique ou virtuel
- c'est lors de la liaison d'adresses (compilation, chargement ou exécution) que les adresses logiques sont traduites en adresses physiques



Concepts

Adresses physiques vs. logiques



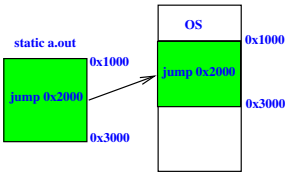
Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison statique

Traduction lors de la compilation ou l'édition des liens

- Résultat : programme absolu
- Le chargement d'un tel programme entraine une demande formulée en termes de taille et d'adresse ce qui laisse peu d'initiatives à l'allocateur
- Si l'adresse de chargement change, il est nécessaire de recompiler
- Cas des programmes .COM de MS-DOS

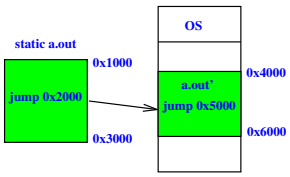


Moment de la traduction d'adresses

UC → @logique → @physique → MC

Liaison au chargement

- résultat : programme translatable
- un chargeur est capable de l'implanter n'importe où en mémoire (réimplantation statique)
- l'adresse du chargement est laissée à l'initiative de l'allocateur
- gestion plus efficace
- limitation : un programme ayant commencé son exécution ne peut plus être déplacé. S'il est swappé sur le disque, il faut qu'il retrouve son emplacement initial (il n'est pas forcément relogeable)



Moment de la traduction d'adresses

UC → @logique → @physique → MC

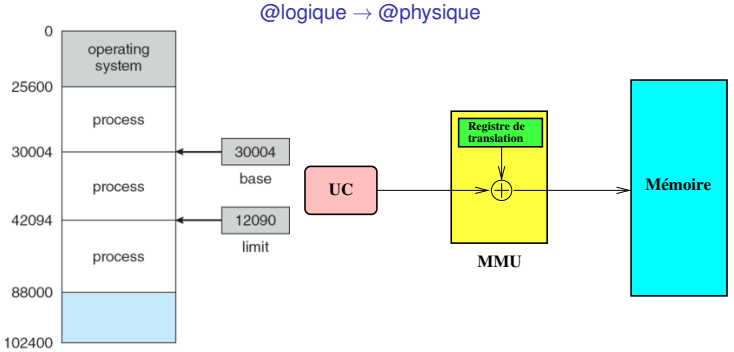
Liaison au moment l'exécution

- traduction dynamique des adresses
- permet la réimplantation dynamique d'un programme au cours de son exécution
- le module résultant est translatable et relogeable
- Implantations possibles (support matériel) :
  - utilisation d'un registre de base (maintien de la contiguité physique du programme)
  - la pagination (sera vue plus tard)

Concepts

Memory Management Unit (MMU)

La MMU est un dispositif matériel qui permet la conversion :
@logique → @physique



Mémoire uniforme vs. mémoire hiérarchisée

Mémoire uniforme

- la mémoire centrale (MC), seule mémoire disponible est organisée linéairement et gérée comme un vecteur d'emplacements contigus
- à chaque processus, est affectée une zone mémoire dont la taille est le maximum estimé de ses besoins
- tout le processus est en mémoire
- Limitations :
  - cas d'un programme dont la taille est supérieure à celle de la mémoire
  - existence de processus inactifs occupant inutilement une partie de la mémoire centrale

Mémoire hiérarchisée (virtuelle)

- la mémoire est constituée d'un ensemble structuré de mémoires de taille et de temps d'accès différents, de telle sorte que les références se fassent toujours à la mémoire la plus rapide
- le cas le plus simple correspond à 2 niveaux : la mémoire centrale et une mémoire secondaire (généralement le disque)

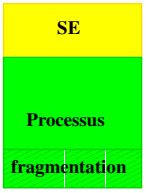
Deuxième partie

Gestion de la mémoire

Mémoire uniforme

Partition unique

- Un seul processus à la fois en mémoire
- Fragmentation interne : espace allouée à un processus mais pas utilisé
- Pour des raisons de protection (SE, processus), on associe au processus un registre de base (de translation) et un registre limite
- C'est le cas de MS-DOS et les systèmes embarqués pour PDA par exemple



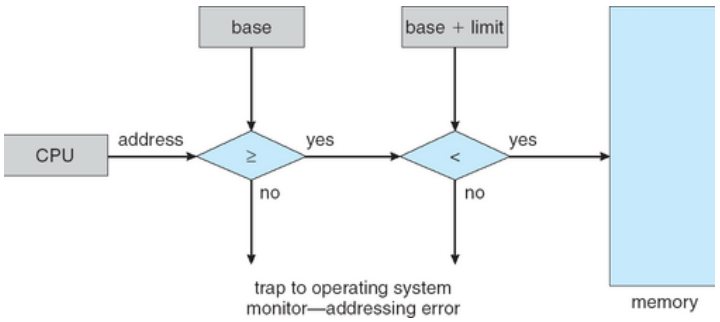
Avantages et inconvénients

- tout le processus est en mémoire
- facile à implanter
- fragmentation interne
- limite le degré de multiprogrammation
- mauvaise exploitation de l'UC et la MC

Mémoire uniforme

Plusieurs partitions de tailles différentes

- allocation contiguë avec un mécanisme matériel de protection
- une partition peut contenir exactement un processus
- les partitions peuvent être fixes ou dynamiques



Mémoire uniforme

Partitions fixes

La taille des partitions est indépendantes des processus chargés

- Avantage : facile à implanter
- Inconvénients :
  - degré de multiprogrammation limité par le nombre des partitions
  - cas des processus dont la taille est supérieure ou inférieure à la taille des partitions ?
  - fragmentation interne
- Solution : partitions dynamiques

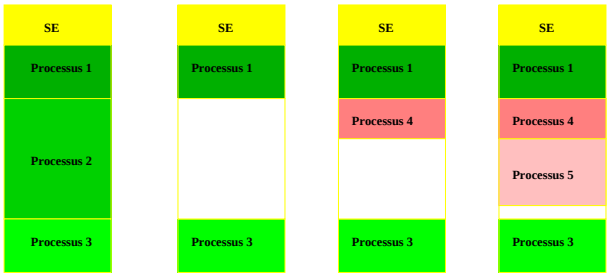
Implantation

```
Charger le processus p dans une partition
Pi libre si T(p) <= T(Pi)
T : taille
```


Mémoire uniforme

Partitions dynamiques

- La taille d'une partition $T(P_i)$ est déterminée en chargeant le programme p_i telle que $T(P_i) = T(p_i)$
- Avantages :
  - augmente le degré de multiprogrammation
  - élimine le problème de la fragmentation interne
- Inconvénient : la fragmentation externe



Mémoire uniforme

Problème de la fragmentation externe

Le compactage est une solution possible si réimplantation dynamique (programmes translatables au moment de l'exécution) :

- compacter tous les processus à chaque terminaison d'un processus :
  - solution chère
- compacter en cas de non possibilité d'allocation pour un processus :
  - compacter tous les processus
  - compacter un processus à la fois jusqu'à satisfaction de la requête
- Inconvénient : le compactage introduit de l'overhead
- une autre solution : la pagination

Mémoire uniforme

Partitions multiples : algorithmes d'allocation

First-fit

- allouer la première partition P_i libre telle que $T(P_i) \geq T(p)$
- Inconvénients :
  - tendence à produire des petits trous au début de la mémoire
  - vitesse de la recherche de plus en plus longue

Best-fit

- allouer la plus petite partition libre telle que $T(P_i) \geq T(p)$
- Inconvénients :
  - recherche coûteuse sur toute la liste des partitions libres
  - fragmentation externe plus importante que les autres algorithmes

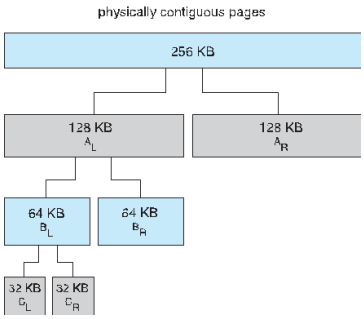
Worst-fit

- allouer la plus grande partition libre telle que $T(P_i) \geq T(p)$
- produit des trous les plus grands possibles

Mémoire uniforme

Algorithme des frères siamois (buddy system)

- Donald KNUTH [1973]
- Liste de blocs libres dont la taille est une puissance de 2 (1, 2, 4, 8 octets, ..., jusqu'à la taille maximale de la mémoire)



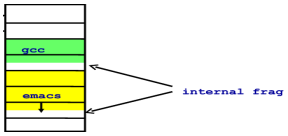
Mémoire uniforme

Pagination

- la mémoire physique est divisée en blocs de taille fixe appelés cadres de pages ou cases
- en termes de mémoire logique, on parle de pages
- la taille des cases est une puissance de 2, généralement entre 512 et 8 192 octets
- la taille d'un cadre est définie par le matériel
- lorsque l'on veut exécuter un programme, on charge ses pages dans les cases disponibles à partir de la mémoire auxiliaire

Remarques

- devoir garder trace de toutes les cases libres
- allocation non contigüe
- une solution à la fragmentation extérieure
- limite le problème de la fragmentation interne

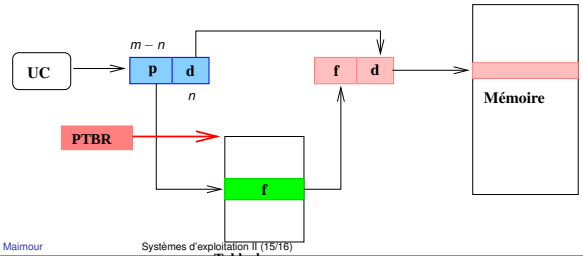


Pagination

Support matériel (MMU)

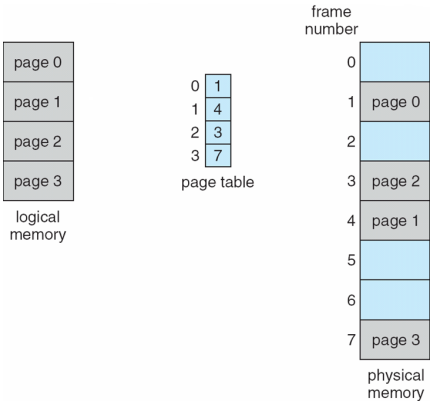
- MMU (Memory Management Unit) : un dispositif matériel qui permet la conversion @logique → @physique
- la table des pages (*TP*) est maintenue en mémoire
- Page-table base register (PTBR) pointe vers la table des pages
- Page-table length register (PRLR) indique la taille de la table des pages
- Machine m bits avec $T(page) = 2^n$

$$(p, d) \rightarrow TP[p] \times T(page) + d$$



Pagination

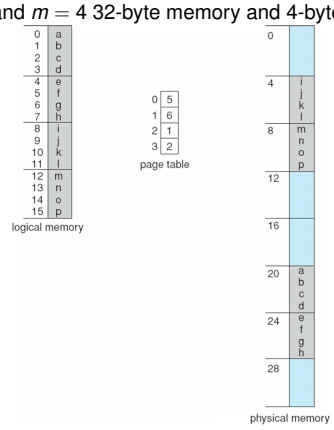
Exemple (1/3)



Pagination

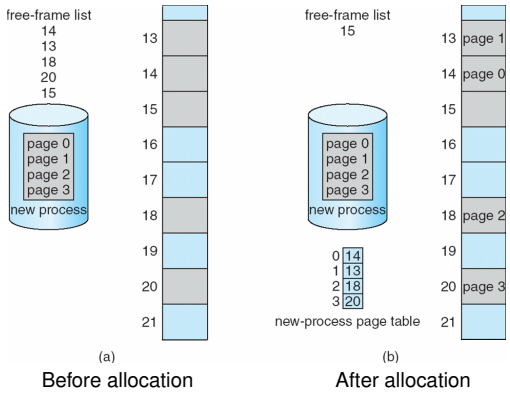
Exemple (2/3)

$n = 2$ and $m = 4$ 32-byte memory and 4-byte pages



Pagination

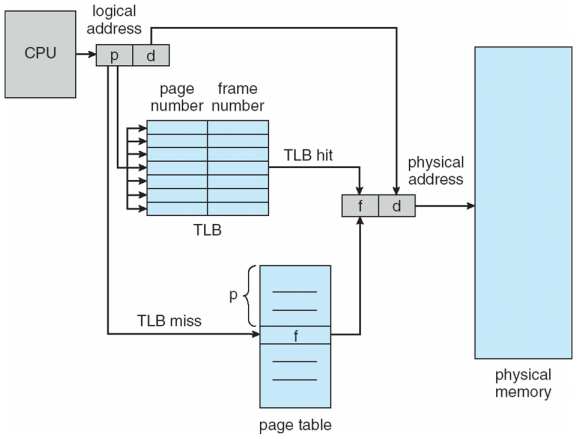
Exemple (3/3) : liste des cadres libres



Translation look-aside buffers (TLBs)

- 2 accès mémoire sont nécessaires pour accéder à une donnée/instruction.
- Solution : utilisation d'une mémoire cache assez rapide appelée mémoire associative ou translation look-aside buffers (TLBs)
- Certaines TLBs assure la protection inter-processus en mémorisant un identificateur pour chaque processus : address-space identifiers (ASIDs)
- Translation $(p, d) \rightarrow (c, d)$:
  si p est disponible dans la TLB, alors récupérer c, sinon aller à la table des pages en mémoire.

TLB



Pagination multiniveaux (1/3)

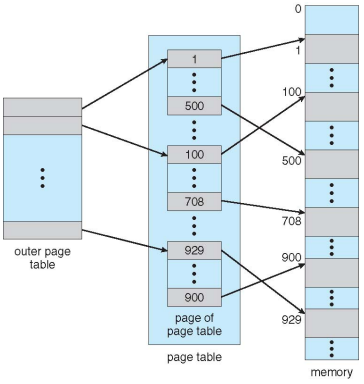
- la table des pages doit être entièrement chargée en mémoire physique.
- Exemple. une machine 32 bits avec des pages de 4K.
  Une adresse logique : numéro de page (20 bits), déplacement (12 bits)
  $\Rightarrow$ taille de la TP est $2^{20} \times 4$ octets
- la pagination multi-niveaux permet de limiter la taille de la table des pages qui doit être résidente en mémoire.
- cas de 2 niveaux, une adresse logique prend la forme :

P1	P2	d
10	10	12

Pagination multiniveaux (2/3)

P1	P2	d
10	10	12

- la mémoire nécessaire pour les différentes TP est de $(2^{10} + 2^{10} \times 2^{10}) \times 4$ octets.
- mais, il n' y a que la TP de niveau 1 qui doit résider en mémoire.
- cas d'un programme de 2^{10} pages, besoin de seulement $2^{10} + 2^{10} = 2^{11}$ entrées résidentes en mémoire



Pagination multiniveaux (3/3)

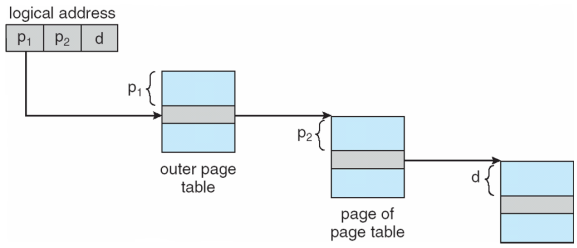


Table des pages inversée

Le problème :

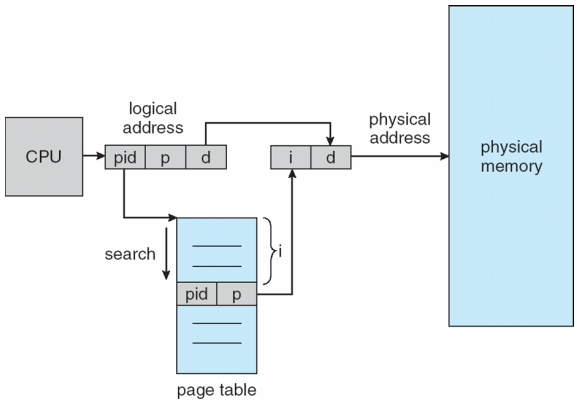
- une machine 32 bits avec une taille de pages de 1K
- une table de pages (associé à chaque processus) a 2^{22} entrées
- comment peut-on réduire cela ?

Solution : table des pages inversée

- une seule table pour tout le système
- une entrée par case (cadre de page) mémoire
- Schéma utilisé par certaines stations de travail IBM et HP
- Inconvénient : parcourir la table pour trouver une correspondance
- Solutions : TLB, hashage.

Pagination

Table des pages inversée



Pagination

Table des pages Hashée

Principe de hashage

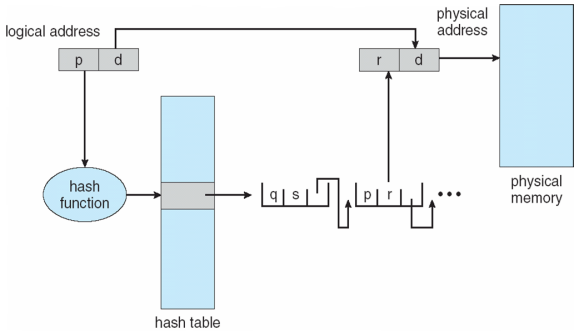
- une fonction surjective qui associe à chaque clé, un ou plusieurs éléments
- au lieu de stocker tous les éléments dans un tableau :
  - une entrée par clé
  - une entrée est une liste de tous les éléments ayant la méme clé
- intérêt : recherche plus rapide en utilisant la clé

Table des pages hashée

- Motivation : réduire la complexité dans les systèmes $>$ 32 bits

Pagination

Table des pages Hashée



Pagination

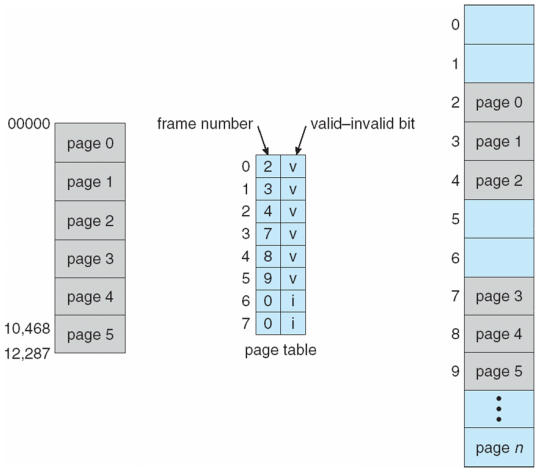
Protection

- Un bit de protection est associé à chaque page mémoire (lecture seule ou lecture/ecriture)
- Un bit (valid/invalid) est associé à chaque page pour indiquer si elle est dans l'espace d'adresse logique du processus

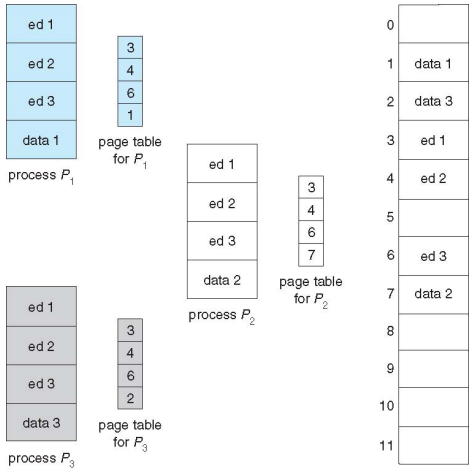
Exemple

- espace d'adresse à 14 bits (0-16383)
- taille d'une page est 2K = 2^{11}
- espace d'adresse = $2^{14}/2^{11} = 8$ pages
- un programme qui n'utilise que les adresses (0-10468)
  - besoin de 6 pages
- les pages 6 et 7 doivent être marquées comme étant invalides

Pagination : la protection



Pagination : le partage



Pagination

Avantages

- pas de fragmentation externe
- séparation (vue utilisateur, mémoire physique)
- protection entre processus implicite
- partage des pages entre plusieurs utilisateurs

Inconvénients

- augmente le temps de commutation de contexte
- fragmentation interne (limitée)

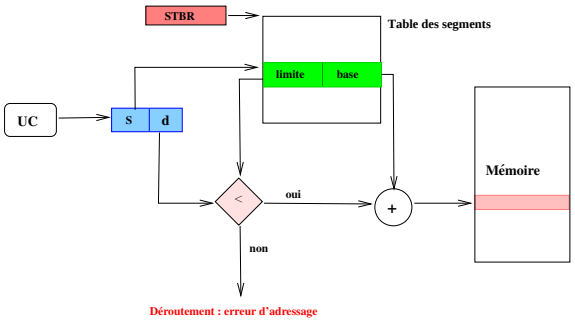
Segmentation

- Allocation non contigüe, partitions de tailles différentes où chaque partition est spécialisée, ce qui permet une vision utilisateur de la mémoire
- Un segment est associé à une portion du programme sémantiquement définie. Exemple :
  - segment code
  - segment données
  - segment pile ...
- Une adresse virtuelle est de la forme <num_segment, déplacement>
- Une table des segments est nécessaire. Elle peut être stockée en mémoire ou dans des registres

Segmentation

Support matériel

- STBR (segment-table base register) : adresse de début de la table des segments
- STLR (segment-table length register) : taille de la tables des segments



Segmentation

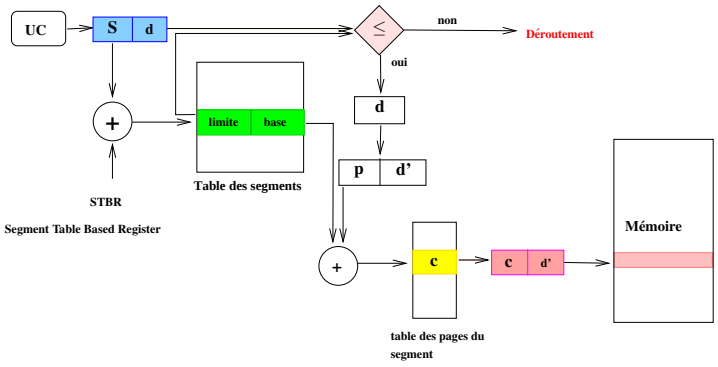
- Protection :
  - entre processus
  - lecture, écriture, exécution
- Partage : possibilité d'utilisation du même segment de code par plusieurs processus différents
- Problème : fragmentation externe quand les trous mémoire sont trop petits pour ranger un segment

Segmentation paginée

- Un segment est composé de plusieurs pages
- À un segment correspond une table de pages : une table de segments et plusieurs tables de pages
- La segmentation combinée avec la pagination permet de réduire la fragmentation externe

Segmentation paginée

Support matériel



Deuxième partie

Gestion de la mémoire

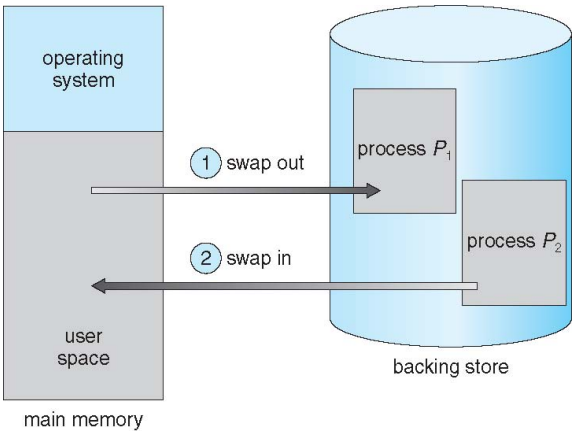
Mémoire virtuelle (hiérarchisée)

- une technique qui permet l'exécution de programmes pouvant ne pas être complètement en mémoire
- la mémoire est vue comme un grand tableau de stockage uniforme : séparation de la mémoire logique (vision utilisateur) de la mémoire physique
- permet une meilleure utilisation de la mémoire : une routine n'est chargée en mémoire qu'au besoin.

Différentes implémentations

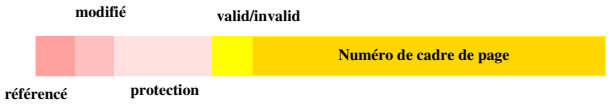
- le swapping
- le swapping à la demande :
  - la pagination à la demande (la plus commune)
  - la segmentation à la demande (difficile à cause de la taille variable des segments).

Swapping



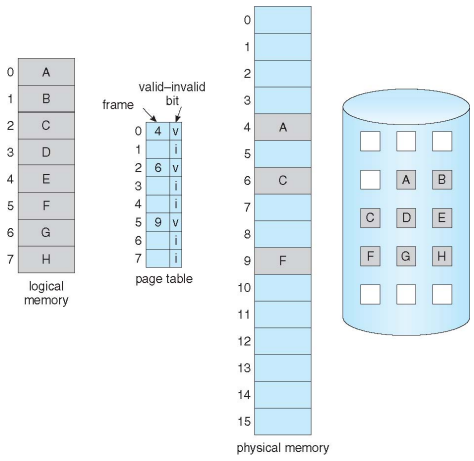
Pagination à la demande

Structure d'une entrée de la table des page



- numéro de cadre de page
- bit de présence (valid/invalid)
- bit (3 bits) de protection
- bit modifié (dirty bit) : si positionné à 1, la page doit être écrite sur le disque (accédée en écriture par un processus)
- bit référencé : mis à 1 à chaque fois que la page est accédée en lecture ou en écriture ⇒ ça sert surtout pour les algorithmes de remplacement de pages

Pagination à la demande



Pagination à la demande

- Une page est transférée de la mémoire virtuelle (MV) à la mémoire centrale (MC) lorsqu'on a besoin de cette page :

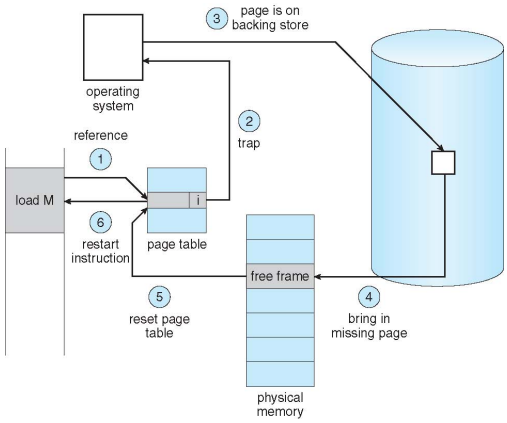
```
loop
   ins(@,opérandes)
   if ins en mémoire then
      exécuter (ins)
   else
      déroutement(défaut de
      page)
   end if
end loop
```

défaut_de_page(*page*)

```
if case libre then
   case ← page
else
   choix (case)
   libérer (case) et y charger (page)
end if
réexécuter (ins)
```

Pagination à la demande

Défaut de page



Pagination à la demande

Charger(p,c)

$TP(p).case \leftarrow c$
$TP(p).bit\_presence \leftarrow 1$

Exécuter(*ins*, *p*)

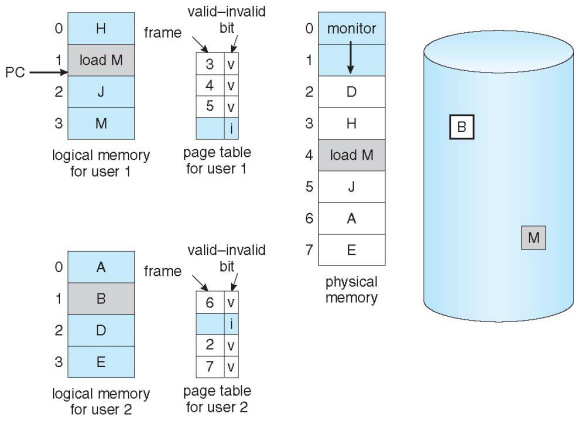
```
if ins modifie p then
    TP(p).bit_modif ← 1
end if
```

Libérer(p)

$TP(p).bit\_presence \leftarrow 0$
`if` $TP(p).bit\_modif = 1$ `then`
  `transférer` *p* `vers la mémoire auxiliaire`
`end if`

Algorithmes de remplacement

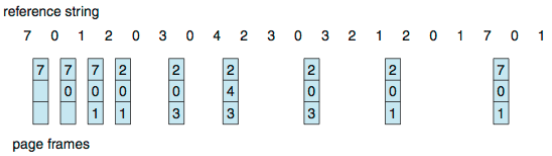
Besoin de remplacement de pages



Algorithmes de remplacement

L'algorithme optimal

- La page victime est celle qui mettra le plus de temps à être référencée.

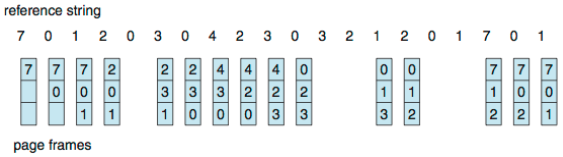


- Algorithme pratiquement impossible.

Algorithmes de remplacement

Premier arrivé, premier servi : FIFO

- la page victime est celle qui a été chargée la première



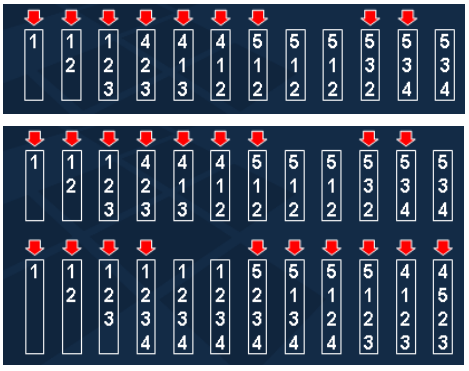
Implémentation

- file d'attente FIFO des pages (la victime en tête de file)

Algorithmes de remplacement

Anomalie de Belady

Soit la chaîne de référence : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.



- 3 cadres de pages : 9 défauts de pages
- 4 cadres de pages : 10 défauts de pages

Algorithmes de remplacement

Not Recently Used : NRU

- associer à chaque pages, 2 bits : R et M initialisés à 0
- à chaque accès en lecture : $R \leftarrow 1$
- à chaque accès en écriture : $M \leftarrow 1$
- à chaque interruption d'horloge, le SE remet R à 0
- Valeurs possibles (RM) :
  - 00 : non modifiée, non référencée
  - 01 : non modifiée, référencée
  - 10 : modifiée, non référencée
  - 11 : modifiée, référencée
- La page victime est choisie au hasard parmi celles ayant le plus petit indice (RM)

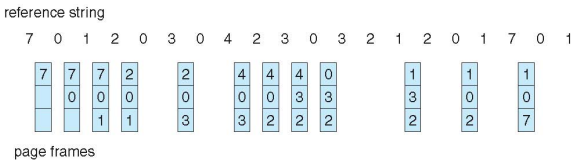
Algorithmes de remplacement

LRU : Least Recently Used

- la page victime est la moins récemment référencée
- **Exemples** :

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	5
2	2	2	2	2
3	5	5	4	4
4	4	3	3	3



Algorithmes de remplacement

LRU : Implémentations

1. Associer à chaque page le moment de sa dernière utilisation.
2. Utiliser une liste chaînée des descripteurs de pages :
   - à chaque référence à une page, la supprimer et la mettre en tête de liste
   - la page victime est celle à la fin de la liste

⇒ algorithme stable mais nécessite une gestion coûteuse de la liste modifiée à chaque accès à une page

Algorithmes de remplacement

Approximations de LRU

Bit de référence (R)

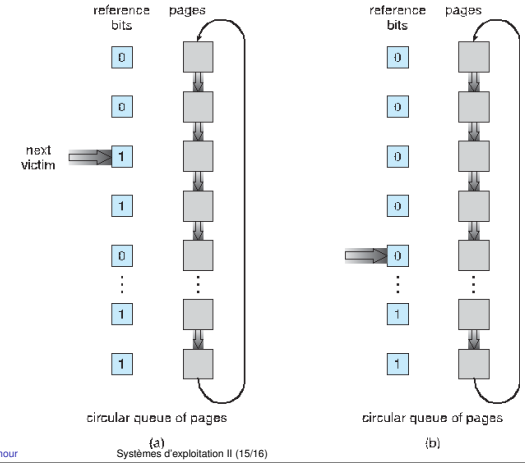
- R est initialement 0 et mis à 1 à chaque référence
- FIFO est appliqué d'abord sur les pages ayant le bit de référence à 0.

La seconde chance

- Le bit de référence est positionné à 1 à chaque référence
- Liste circulaire avec un pointeur sur la prochaine victime
- Si la page à remplacer est telle que R = 0, on la remplace sinon on lui donne une 2ème chance :
  - $R \leftarrow 0$
  - laisser la page en mémoire
  - passer à la suivante dans la liste en appliquant les mêmes règles.

Algorithmes de remplacement

La seconde chance



Algorithmes de remplacement

Autres algorithmes

- LFU (Least Frequently Used) : la victime est la page la moins fréquemment utilisée
- MFU (Most Frequently Used) : la victime est la page la plus fréquemment utilisée
- Implémentation : associer à chaque page, un compteur contenant le nombre de ses références
  - LFU : remplacer la page avec le compteur minimum
  - MFU : remplacer la page avec le compteur maximum
- Remarque : LFU et MFU sont difficiles à implanter

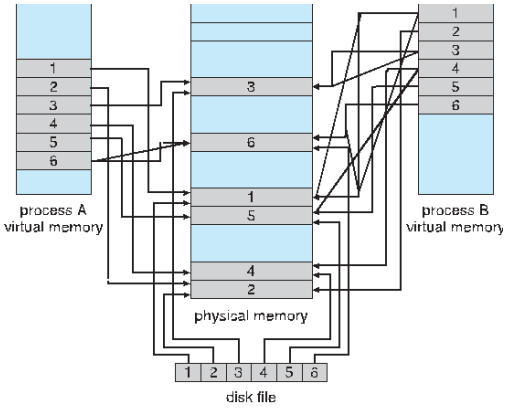
Mémoire virtuelle

Allocation de cadres de pages

- En fonction de l'architecture, un minimum de cadres de pages doit être alloué à chaque processus.
- Algorithmes d'allocation
  - allocation fixe à parts égales ou proportionnelles à la taille des processus
  - allocation avec priorité :
    - si défaut de page (P_1) alors remplacer une page d'un processus P_2 moins prioritaire
- Allocation globale vs locale
- L'écroulement (thrushing)
- Prépagination

Mémoire virtuelle

Fichiers mappés en mémoire
(Memory mapped files)



Etude de cas

Exemples d'architectures

- SPARC (32 bits) pagination à 3 niveaux
- Motorola 68030 (32bits) à 4 niveaux
- VAX segmentation paginée (2 niveaux) machine 32 bits
- Intel Pentium

Etude de cas

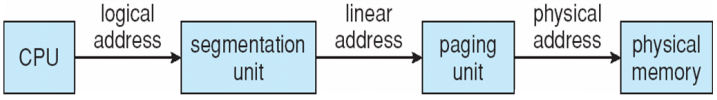
Intel Pentium (IA-32)
Adresses mémoire

- Adresse logique : utilisée par le programme pour spécifier l'adresse d'un opérande ou d'une instruction.
  adresse logique = segment , offset
- Adresse linéaire : un entier non signé sur 32 bits permettant l'adressage de 4GB.
  de 0x0000 0000 à 0xffff ffff
- Adresse physique : l'ensemble des signaux envoyés par le processeur à la mémoire sur le bus d'adresse permettant l'accès à une cellule mémoire.
- MMU (Memory Management Unit) : un dispositif matériel qui permet la conversion @logique → @physique

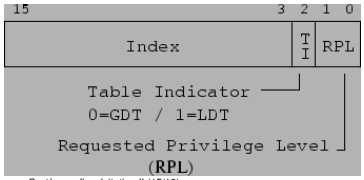
Etude de cas

Intel Pentium (IA-32)

- supporte à la fois la segmentation et la segmentation paginée
- l'UC génère des adresses logiques transformées par l'unité de segmentation en adresses linéaires
- les adresses linéaires sont transformées par l'unité de pagination en adresses physiques.



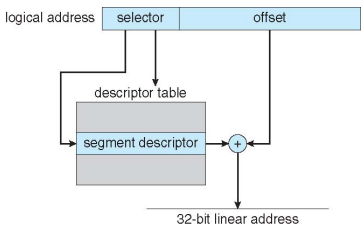
- @logique = selector (16 bits) + offset (32 bits)



Etude de cas : Intel Pentium (IA-32)

Unité de segmentation

- Descripteur de segment (8 octets) :
  - base (32 bits) : @linéaire du 1ér octet du segment
  - limit (20 bits) : longueur du segment en octets (G = 0) ou en pages (G = 1)
- GDT Global Descriptor Table (pour les segments système)
- LDT Local Descriptor Table (par processus)
- GDTR Registre pointant sur la GDT
- LDTR Registre pointant sur la LDT en cours



Adresse linéaire

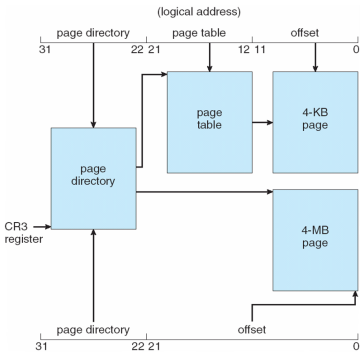
page number		page offset
p_1	p_2	d
10	10	12

Etude de cas : Intel Pentium (IA-32)

Unité de pagination

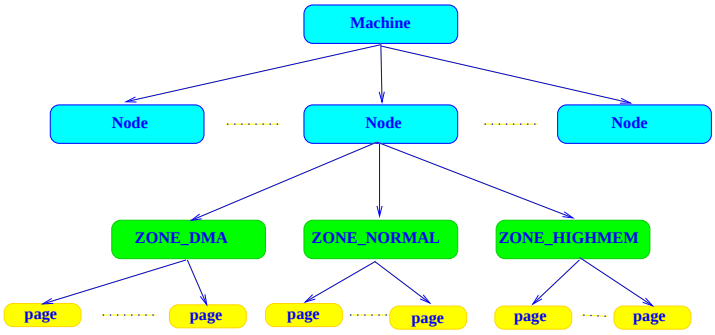
- la pagination est activée : flag PG du registre cr0
- taille de page : 4K
- cr3 contient l'@physique de la "page directory"

- Pagination étendue : pages de taille 4K ou 4M
  - activée : flag "Page Size" de l'entrée de la page directory
  - @linéaire = page (10bits) + offset (22 bits)



Etude de cas : Linux

Organisaion physique de la mémoire



Etude de cas : Linux

Organisaion physique de la mémoire

- Machine 32 bits (Pentium)
- l'espace d'adressage d'un processus = 2^{32} = 4Go :
  - 3 Go pour le processus
  - 1 Go réservé et utilisé par le processus en mode noyau : table des pages et autres données du noyau
- l'espace d'adressage est divisé en zones homogènes et contiguës

zone	physical memory
ZONE_DMA	< 16 MB
ZONE_NORMAL	16 .. 896 MB
ZONE_HIGHMEM	> 896 MB

Etude de cas : Linux

Espace d'adressage d'un processus : régions

- une région contient un ensemble de pages consécutives possédant les mêmes propriétés de protection : région de code, région de fichiers mappés . . .
- à un processus, est associée une liste de descripteurs de régions (vm_area_struct)
  - **vm_mm** : la mm_struct à laquelle appartient cette région.
  - **vm_start, vm_end** : l'adresse de début et fin de la région.
  - **vm_next** : toutes les régions d'un processus sont reliées entre elles dans l'ordre de leurs adresses par **vm_next**.
  - **vm_page_prot** : les flags de protection positionnés dans chacune des entrées de la table des pages de cette région.
  - **vm_flags** : un ensemble de flags décrivant les propriétés et la protection de la région :
    - VM_READ
    - VM_WRITE
    - VM_EXEC
    - VM_SHARED
    - . . .

Etude de cas : Linux

Espace d'adressage d'un processus : régions

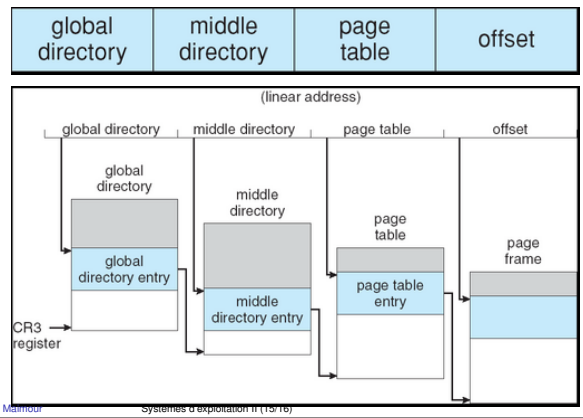
cat /proc/self/maps

```
08048000-0804c000 r-xp 00000000 03:07 293186 /bin/cat
0804c000-0804d000 rw-p 00003000 03:07 293186 /bin/cat
0804d000-0806e000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:07 390929 /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:07 390929 /lib/ld-2.3.2.so
40017000-40018000 rw-p 00000000 00:00 0
40027000-4014f000 r-xp 00000000 03:07 390935 /lib/libc-2.3.2.so
4014f000-40157000 rw-p 00127000 03:07 390935 /lib/libc-2.3.2.so
40157000-4015a000 rw-p 00000000 00:00 0
4015a000-401a9000 r--p 00000000 03:08 734469 /usr/lib/locale/locale-archive
bfffe000-c0000000 rwxp fffff000 00:00 0
```


Etude de cas : Linux

Pagination à 3 niveaux

- Taille d'une page mémoire est 4Ko
- Pagination à 3 niveaux généralisée à toutes les architectures



Etude de cas : Linux

- Le noyau est totalement résident en mémoire
- Le reste de la mémoire : mémoire dynamique
- Buddy algorithm pour l'allocation de mémoire contiguë
- Pagination à la demande.

Etude de cas : Linux

Remplacement de pages

- Tenter de garder un minimum de pages libres
- Au départ, init lance un démon de pagination kswapd qui s'exécute toutes les secondes
- kswapd vérifie le nombre de pages libres, si bon, il se rendort
- sinon il lance 3 procédures de récupération de pages :
  - cache de pagination ou le tampon du cache
  - partagées qu'aucun processus ne semble utiliser abondamment
  - utilisateur ordinaires
- bdflush se réveille périodiquement. Il vérifie si une trop forte proportion de pages est modifiée : écriture sur disque

Etude de cas : Linux

Remplacement de pages : kswapd