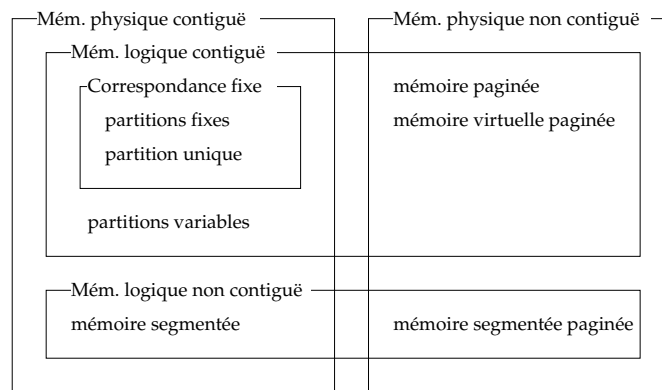


►► Allocation de la mémoire centrale ◀◀

- Généralités,
- Correspondance statique,
 - Partition unique,
 - Partitions fixes,
- Correspondance dynamique,
 - Partitions variables,
 - Allocation de partitions variables,
 - Fragmentation externe/interne,

1

►► Catégories de mémoire ◀◀



2

Notion de mémoire : La *mémoire* est une ressource

- partageable / non partageable,
- réquisitionnable / non réquisitionnable,
- réutilisable.

Chaque processus travail dans une *mémoire logique* qui est vu comme un tableau de cases mémoire.

Cette mémoire logique est dite *linéaire* car les cases qui la composent sont contiguës. Ces cases sont repérées par des entiers consécutifs qui sont appelés des *adresses logiques*.

Lors de l'exécution, chaque processus P génère des adresses logiques.

A l'opposé, les cases de la mémoire physique sont repérées par des *adresses physiques*.

3

Correspondance entre adresses logiques et adresses physiques,

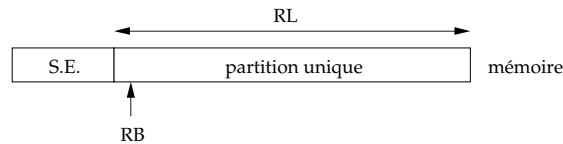
- *statique* (établie une seule fois),
- *dynamique* (variable dans le temps).

Gestion de la mémoire physique.

Partage de données entre processus.

Protection de chaque processus.

►► Système à partition unique ◀◀



Le *va-et-vient* simple ou *swapping*. Caractéristiques :

- les processus sont tous et toujours à la même adresse physique,
- la correspondance logique/physique est réglée lors de la compilation ou du chargement.
- la réquisition de la CPU entraîne la sauvegarde de la partition et le chargement d'un nouveau processus,
- la CPU est inutilisée durant les sauvegardes/récupérations.

La protection est possible par le jeu des deux registres spécialisées RB (registre de base) et RL (registre limite).

Pour limiter les E/S, sauvegarder uniquement les zones de la partition qui ont été modifiées et/ou appliquer l'*algorithme des pelures d'oignons*.

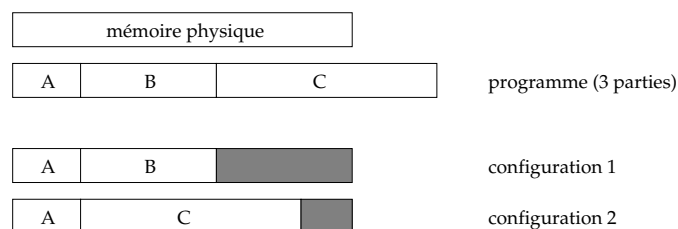
5

►► Les recouvrements ◀◀

Objectif : Limiter la taille de la mémoire occupée par le code du programme.

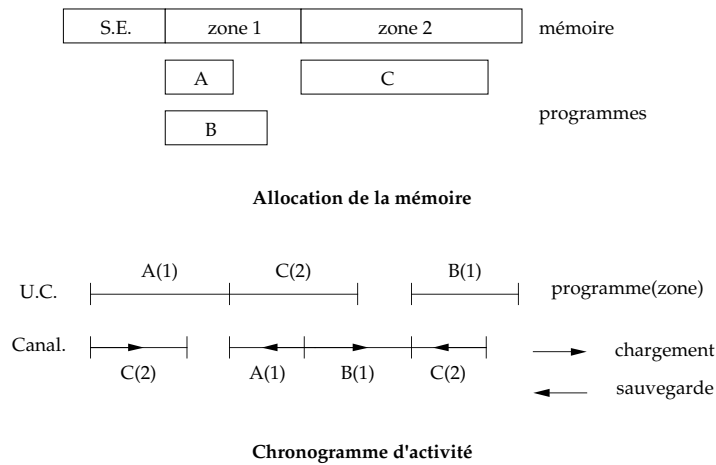
Caractéristiques :

- Les programmes importants sont découpés en plusieurs parties *indépendantes*.
- Une partie reste systématiquement en mémoire, c'est la *racine du recouvrement*.
- Les autres parties sont chargées en mémoire à la demande.



6

►► Système à partitions fixes ◀◀



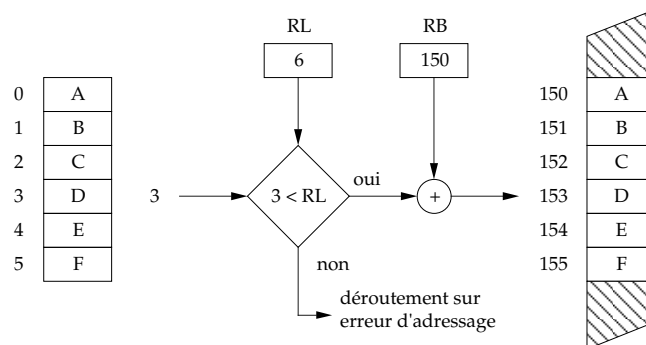
- La mémoire est divisée en zones de taille fixe (spécialisation).
- Les processus s'exécutent toujours dans la même zone.
- pendant les échanges sur la zone 1, un autre processus s'exécute sur la zone 2.
- la correspondance logique/physique est réglée lors de la compilation ou du chargement.
- Protection possible par les registres RB et RL.

7

►► Système à partitions variables ◀◀

Les partitions sont **allouées et libérées à la demande** (création ou fin d'un processus).

Le registre de base pointe sur la partition et le registre limite en indique la taille.



Les processus génèrent des adresses logiques comprises entre 0 et RL-1. RB et RL sont utilisés par la CPU pour traduire ces adresses en adresses physiques à chaque accès mémoire.

8

►► Allocation par chaînage des zones libres ◀◀

Les **zones libres** sont placées dans une liste chaînée. Il existe plusieurs **stratégies** de recherche :

- Première zone libre (*first-fit*).
- Meilleur ajustement (*best-fit*) pour utiliser au mieux les zones libres.
Mais,
 - il faut parcourir toutes les zones,
 - il reste des petits résidus inutilisables.
- Plus grand résidu (*worst-fit*) pour combattre l'émiettement.
- Algorithmes spécialisés.

9

►► Allocation par subdivision (buddy system) ◀◀

La taille des zones suit une règle définie par une relation de récurrence :

- binaire : (1, 2, 4, 8, ...) $S_{i+1} = 2 \times S_i$
- fibonacci : (1, 2, 3, 5, 8, ...) $S_{i+1} = S_i + S_{i-1}$

Il existe une liste des zones libres pour chaque taille.

```
⟨allouer un bloc de taille T⟩
|
| si ⟨il existe un bloc de taille T⟩ alors
|   | ⟨renvoyer ce bloc⟩
| sinon
|   | ⟨allouer un bloc de taille 2 × T⟩
|   | ⟨libérer la deuxième partie de ce bloc⟩
|   | ⟨renvoyer la première partie⟩
| fin si
```

La recherche d'une zone libre est rapide et il est facile de reconstruire des zones à la libération.

10

Un exemple d'allocations/libérations :

8					
A	2	4			Alloc. de A, longueur = 2
A	B	1	4		Alloc. de B, longueur = 1
A	B	1	C	2	Alloc. de C, longueur = 2
2	B	1	C	2	Libération de A
2	B	1	4		Libération de C
8					Libération de B

11

►► Fragmentation externe/interne ◀◀

La *fragmentation externe* est due à l'émiettement de la mémoire lors des allocations/libérations.

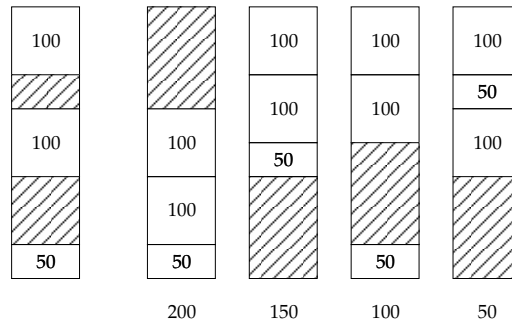
La *fragmentation interne* c'est l'unité de mémoire minimum que le S.E. est capable de gérer (généralement plusieurs Kilooctets).

12

►► Compactage de la mémoire ◀◀

Le compactage de la mémoire procède par *recopie* des partitions pour faire apparaître une zone libre de taille suffisante.

Exemple :



13

►► Mémoire paginée ◀◀

- Principe,
- Correspondance des adresses,
- Mémoires associatives,
- Partage de pages,

14

►► Mémoire paginée : principe ◀◀

La mémoire (logique et physique) est divisée en *page de taille fixe* (quelques Kilo-octets). Cette *taille* est toujours une puissance de deux (2^l).

Une *adresse logique* (sur n bits) dans un système paginée est un couple

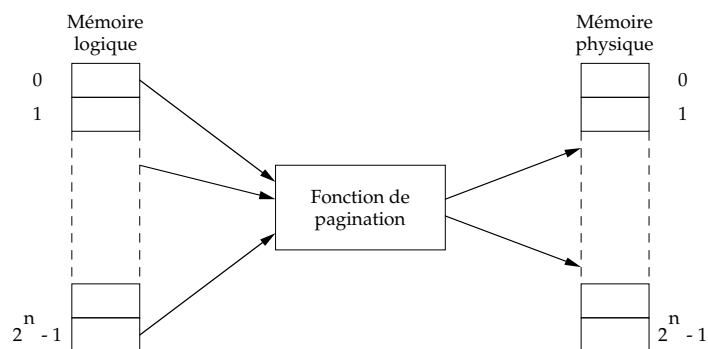
$\langle n^\circ \text{ de page logique (sur } n - l \text{ bits), déplacement (sur } l \text{ bits)} \rangle$

Une *adresse physique* (sur m bits) est un couple

$\langle n^\circ \text{ de page physique (sur } m - l \text{ bits), déplacement (sur } l \text{ bits)} \rangle$

15

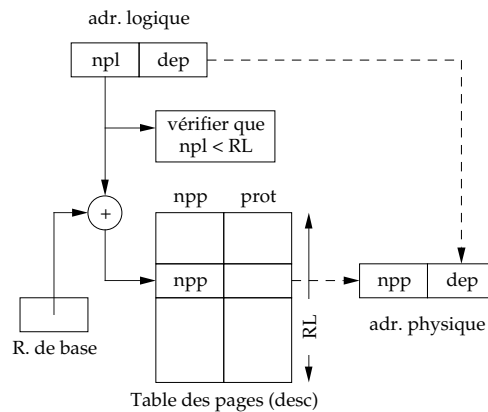
La **fonction de pagination** assure la correspondance entre numéro de page logique et numéro de page physique.



16

▶▶ Pages logiques versus pages physiques ◀◀

Pour chaque processus, le S.E. détient une *table de pages logiques* (notée desc).



17

Version algorithmique de la correspondance :

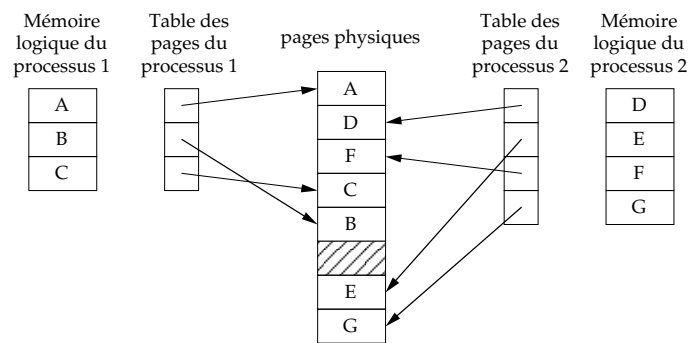
```

⟨npl, dépl⟩ := adresse logique
si (npl < RL) alors
    | si les protections desc[npl].prot sont respectées alors
    |   | npp := desc[npl].npp
    |   | adresse_physique := ⟨npp, dépl⟩
    | sinon
    |   | ⟨déroutement sur violation de protection⟩
    | fin si
sinon
    |   | ⟨déroutement sur erreur d'adressage⟩
fin si
    
```

18

►► Exemple et discussion ◀◀

Un exemple avec deux processus :



Avantages :

- La gestion de la mémoire est plus simple (il suffit de gérer la liste des pages libres).
- Le compactage est inutile.
- Protections différentes pour chaque page.

Inconvénients :

- temps d'accès = $2 \times t = 2 \times 100$ nanosecondes.
- Il faut que le matériel supporte cette organisation.

19

►► Comportement des processus ◀◀

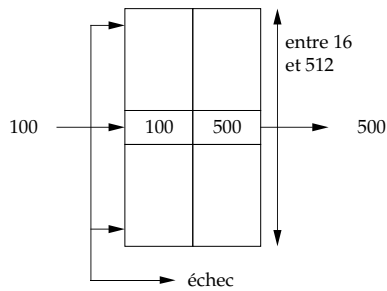
Comportement « en moyenne » des processus :

- *Non uniformité* : 20% des pages regroupent 75% des accès.
- *Principe de localité* :
 - stabilité des accès sur une *courte période*,
 - l'activité actuelle est une *bonne estimation* de l'activité future.

20

▶▶ Mémoires associatives ◀◀

Principe des mémoires associatives :

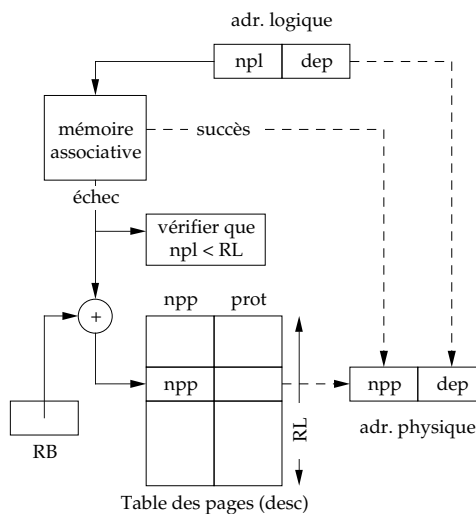


- Peu de temps d'attente car les tests sont faits en parallèle (≈ 20 nanosecondes).
- Ces circuits sont très onéreux.

21

▶▶ Mémoires associatives et pagination ◀◀

Principe : Retenir les derniers couples (page logique, page physique), pour éviter l'accès mémoire à la table des pages.



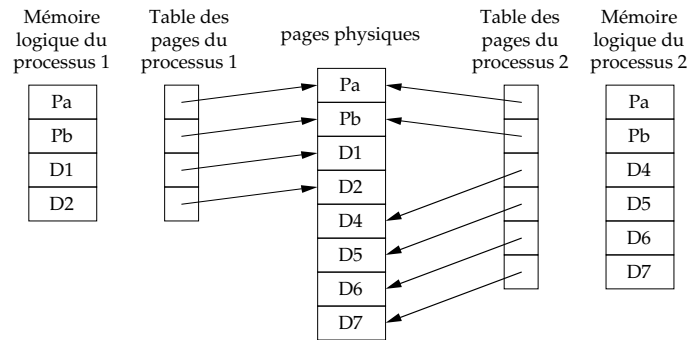
Il faut mettre à jour la M.A. après les échecs et la vider lors des commutations.

Le taux de réussite est lié à la taille de la M.A. (entre 80% et 95%).

- $0,80 \times (100 + 20) + 0,20 \times (100 + 100 + 20) = 140$ ns
- $0,95 \times (100 + 20) + 0,05 \times (100 + 100 + 20) = 125$ ns

22

►► Partage de pages entre processus ◀◀



Les pages contenant le programme (Pa et Pb) sont partagées, mais les pages de données (D1, ..., D7) ne le sont pas.

Les pages contenant le programme Pa et Pb sont *partagées*, tandis que les pages Dx ne le sont pas.

Pour une même page physique, il est possible d'avoir des protections différentes suivant le processus qui l'utilise.

23

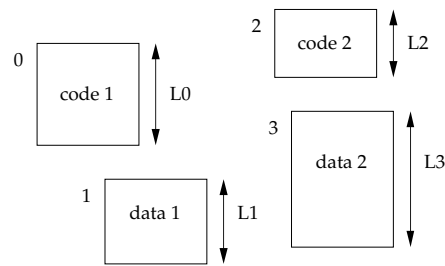
►► Mémoire segmentée ◀◀

- Notion de segments,
- Correspondance des adresses,
- Partage de segments,
- Pagination d'une mémoire segmentée,

24

►► Notion de segments ◀◀

Un *segment* est un bloc de donnée de *taille variable*.



Une *adresse logique* dans un système segmenté (aussi appelée *adresse segmentée*) est un couple

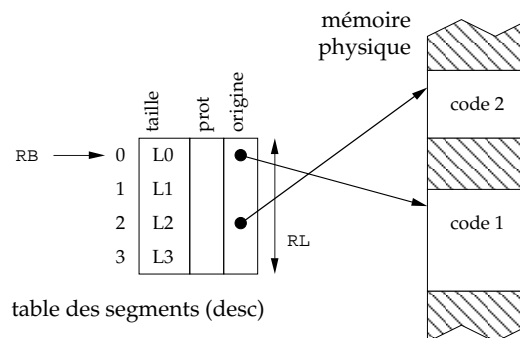
⟨ numéro de segment, déplacement ⟩

Cette organisation physique de la mémoire correspond bien à l'organisation logique à l'intérieur d'un programme.

25

►► Transformation des adresses segmentées ◀◀

Le S.E. détient une *table des segments* pour chaque processus.



Correspondance :

```
⟨seg, depl⟩ := adresse logique
si (seg < RL) et (depl < desc[seg].taille) alors
  | si les protections desc[seg].prot sont respectées alors
  | | adresse physique := desc[seg].origine + depl
  | sinon
  | | ⟨déroutement sur violation de protection⟩
  | fin si
sinon
  | ⟨déroutement sur erreur d'adressage⟩
fin si
```

26

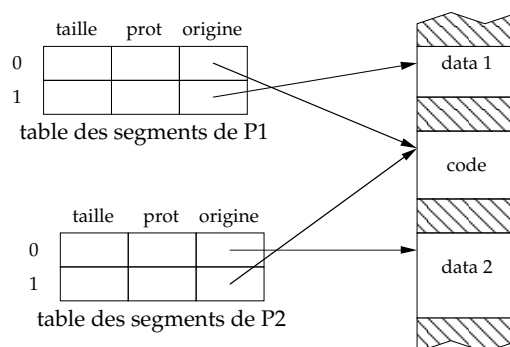
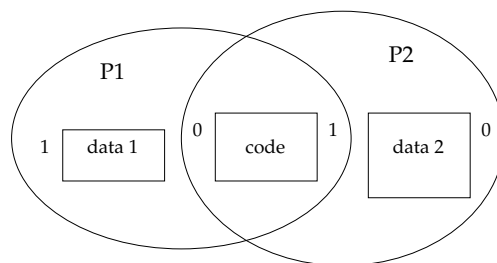
Inconvénients : l'allocation de segments implique

- un problème de fragmentation externe et
- une obligation du tassage de la mémoire.

Avantages :

- les protections concernent les segments,
- on peut utiliser les mémoires associatives.
- le partage de segment est simple :

27

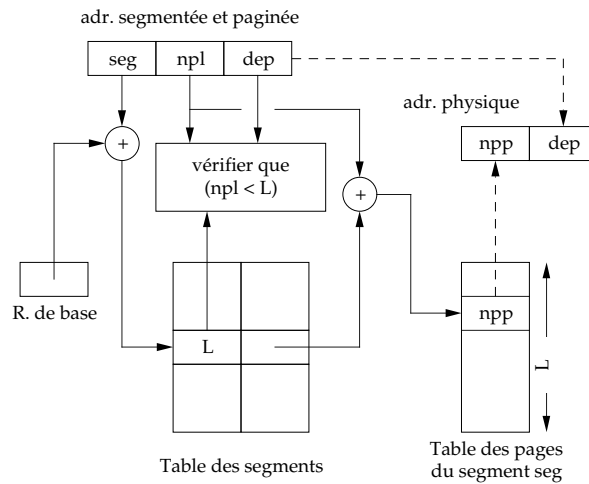


28

►► Pagination d'une mémoire segmentée ◀◀

Une *adresse logique* (ou *adresse segmentée et paginée*) est un triplet
(numéro de segment, numéro de page, déplacement)

La *taille des segments* s'exprime maintenant en nombre de pages.



Avantages : plus de problème de fragmentation interne puisque les segments ne sont plus contigus en mémoire physique.

►► Mémoire virtuelle paginée : principe ◀◀

principe : les programmes utilisent 20% de leur page, donc il est inutile de toutes les conserver en mémoire.

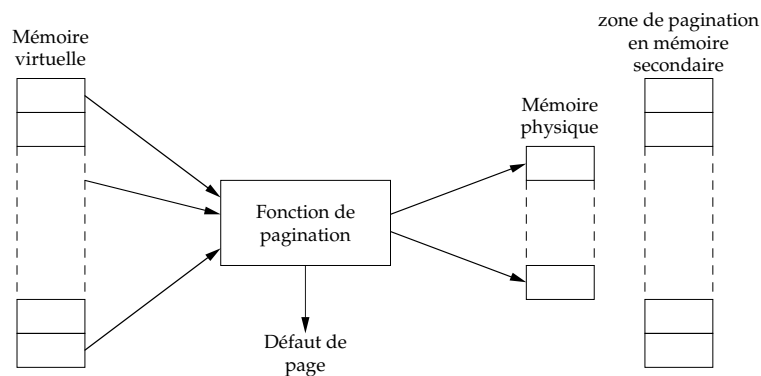
Exemple : 1000 pp = 10 processus de 100 pages logiques ou 50 processus de (100 × 0,2) pages utiles.

Le S.E. doit détecter (avec l'aide du matériel) :

- les pages inutilisées (réquisition),
- les pages utiles et présentes en mém. phys.,
- les pages utiles et absentes de la mémoire physique (*défaut de page*).
- les pages utiles dans le futur et absentes de la mémoire physique (*préchargement*).

1

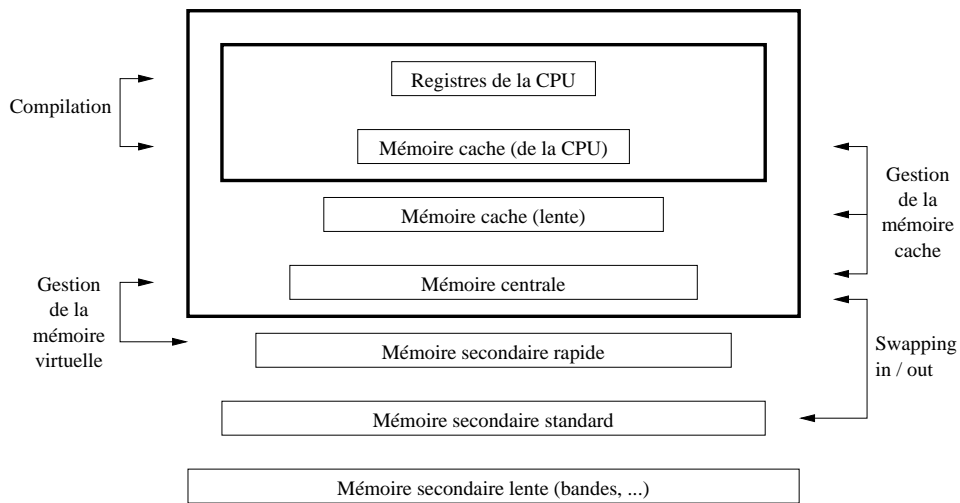
Fonction de pagination virtuelle :



2

►► Hiérarchie de mémoire ◀◀

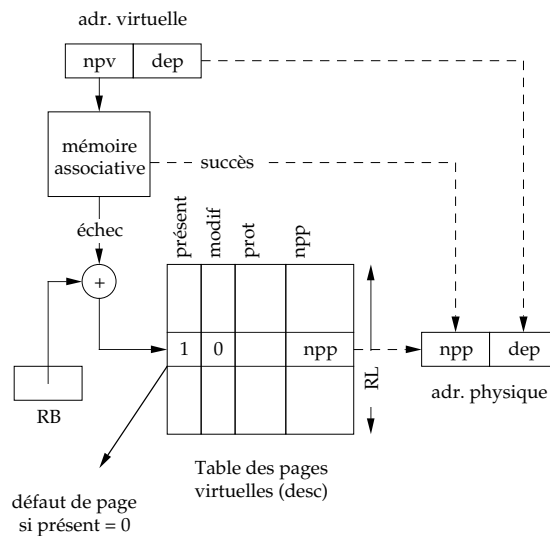
La mémoire virtuelle implante la gestion d'un cache :



3

►► Adresses virtuelles versus adresses physiques ◀◀

Pour chaque processus, le S.E. détient une table des pages virtuelles (pointée par le registre de base) :



4

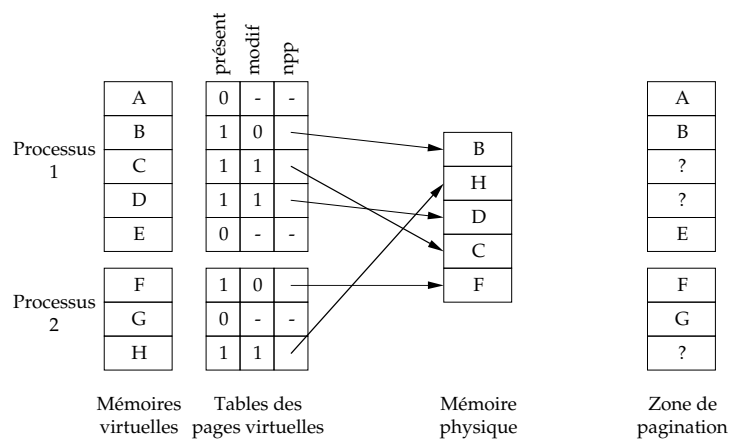
Correspondance des adresses (c'est la partie matérielle de la pagination) :

```

⟨npv, depl⟩ := adresse_virtuelle
⟨vérifier que (npv < RL)⟩
⟨vérifier le respect des protections desc[npv].prot⟩
si (desc[npv].présent = 1) alors
    | npp := desc[npv].npp
    | adresse_physique := ⟨npp, depl⟩
sinon
    | ⟨déroutement sur défaut de page⟩
fin si
    
```

5

►► Un exemple sur deux processus ◀◀



6

►► Le traitement du défaut de page ◀◀

Notations :

- page[i] : la page **physique** de numéro i ,
- swap[j] : la page de numéro j dans la zone de pagination.

Algorithme :

```
⟨suspendre le processus qui a provoqué le défaut⟩
si il existe une page physique libre alors
  | npp := ⟨numéro de cette page⟩
sinon
  | v := ⟨choisir un page virtuelle victime⟩
  | npp := desc[v].npp
  | desc[v].présent = 0 ;
  | si (desc[v].modif = 1) alors
  | | swap[v] := page[npp]
  | | c'est une E/S
  | fin si
fin si
npv := ⟨numéro de la page virtuelle manquante⟩
page[npp] := swap[npv]
desc[npp].présent := 1
desc[npv].modif := 0
desc[npv].npp := npp
⟨reprendre le processus qui a provoqué le défaut⟩
```

7

►► Pagination a plusieurs niveaux ◀◀

Principe : Si la mémoire est importante le nombre de pages augmente et la table des pages devient imposante.

Exemple : Une mémoire de 256 Mo (soit 2^{28} octets) est divisée en $2^{28}/2^{10} = 2^{18}$ pages. La table des pages a donc 2^{18} entrées soit 1 Mo.

Solution : paginer la table des pages ce qui revient à faire une pagination à deux niveaux.

8

►► Discussion sur la taille des pages ◀◀

La taille des pages doit être **grande** pour

- diminuer le nombre de pages, donc le nombre de défauts de page et la taille de la table des pages ;
- optimiser le temps de transfert vers ou depuis la zone de pagination ;
- utiliser des mémoires centrales de plus en plus grandes ;

La taille des pages doit être **petite** pour

- limiter la fragmentation interne ;
- définir avec plus de précision les zones de la mémoire utiles à un processus.

Actuellement la taille des pages varie entre 1 ko et 32 ko.

Certains systèmes autorisent plusieurs tailles différentes.

11

►► Algorithmes de remplacement ◀◀

On choisit en priorité les *pages virtuelles propres* (qui n'ont pas été modifiées).

- Algorithme optimale (base de référence) : choisir la page virtuelle qui est utilisée le plus tard possible ou qui n'est plus utilisée.
- Algorithme aléatoire (le moins bon)..
- Algorithme FIFO (il ne tient pas compte de l'utilisation des pages).
- Algorithme LRU (*Least Recently Used*) est basé sur le principe de localité : choisir la page dont la date du dernier accès est la plus ancienne.
- Algorithme LFU (*Least Frequently Used*) : choisir la page la moins utilisée.

12

►► Algorithmes de remplacement (suite) ◀◀

Algorithme FINUFO (*First In Not Used First Out*) ou algorithme de la *deuxième chance*. On dispose

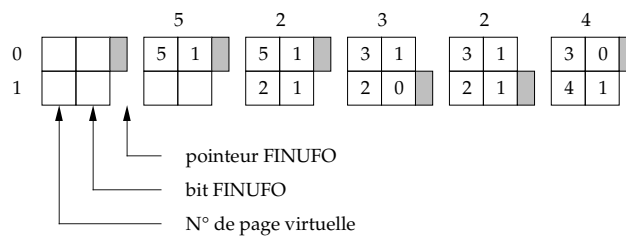
- d'un pointeur de page physique ptr ,
- d'un bit par page physique noté $U[k]$. Ce bit $U[k]$ est forcé à 1 après chaque accès à la page physique k .

Algorithme du choix de la victime FINUFO :

```
tant que ( $U[ptr] = 1$ ) faire  
  |  $U[ptr] := 0$   
  |  $ptr = (ptr + 1) \bmod \langle \text{nb de pages physiques} \rangle$   
fin faire  
 $U[ptr] = 1$   
 $ptr = (ptr + 1) \bmod \langle \text{nb de pages physiques} \rangle$ 
```

13

Un exemple pour une mémoire physique à deux pages :

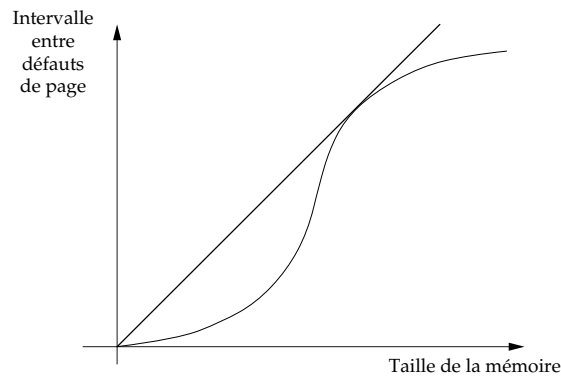
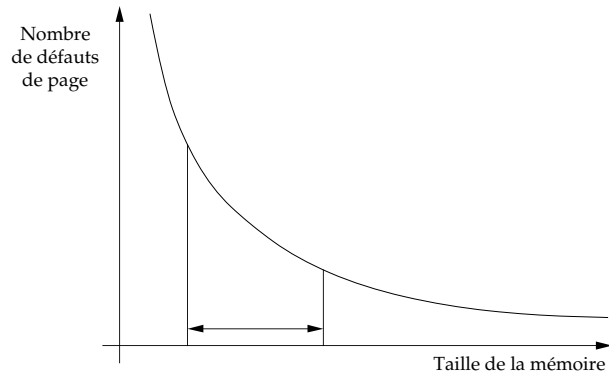


Performances :

OPT > LRU > LFU > FINUFO > FIFO > ALEA

14

►► Comportement en mém. virt. paginée ◀◀



15

►► Allocation des pages physiques ◀◀

Allocation équitable : On donne à chaque processus une part égale de la mémoire physique.

Allocation proportionnelle : on donne à chaque processus une part proportionnelle à la taille de sa mémoire virtuelle.

	Virtuelles	Physiques
Processus 1	25	20
Processus 2	70	53
Processus 3	35	27
Mémoires	130	100

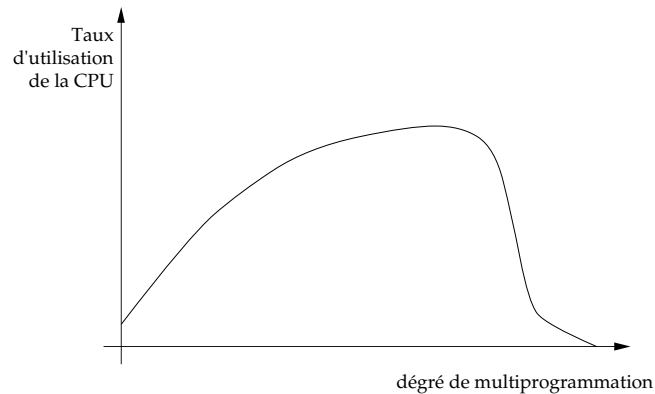
Politique de choix de la victime :

- *Remplacement local* : on choisit parmi les pages du processus demandeur.
- *Remplacement global* : on choisit parmi toutes les pages.

16

►► Écroulement d'un système paginé ◀◀

Définition : Augmentation considérable du nombre de défaut de page provoquant un chute du taux d'utilisation de la CPU.



17

Pour éviter l'écroulement on utilise

- la *régulation de charge par variation du degré de multiprogrammation (swapping in/out)* ou
- la *répartition variable* de la mémoire physique ;

en se basant sur

- la *méthode de l'ensemble de travail*,
- l'observation du *taux de défauts de page*.

18

L'ensemble de travail d'un processus au temps t noté $W(t, T)$ est l'ensemble des pages ayant été référencées entre t et $(t - T)$. T est appelé la *fenêtre d'observation*.

Propriété : la probabilité que l'une des pages de $W(t, T)$ soit référencée au temps $(t + 1)$ est forte.

Conséquence : si les pages physiques allouées à un processus ne peuvent contenir l'ensemble de travail, le nombre de défauts de page sera important.

Problème : il est très coûteux de maintenir un ensemble de travail pour chaque processus.

19

On utilise une approximation en associant n bits b_1, \dots, b_n à chaque page physique.

- chaque accès à une page physique provoque la mise à 1 du bit b_1 associé à cette page ;
- régulièrement (sur interruption d'horloge), le S.E. décale les bits b_1, \dots, b_{n-1} vers la droite et le bit b_1 est forcé à zéro ;

donc, les bits b_1, \dots, b_n donnent un historique d'utilisation de la page physique.

une page physique appartient à $W(t, T)$ ssi il existe $j < T$ tel que $b_j = 1$.

►► La méthode du taux de défaut de page ◀◀

Le taux de défaut de page pour chaque processus doit être *compris entre deux bornes*.

Si le taux est **trop bas**, alors le nombre de pages physiques allouées à ce processus est trop important.

Si le taux est **trop haut**, alors le nombre de pages physiques allouées à ce processus n'est pas suffisant pour contenir son ensemble de travail. Deux actions sont possibles :

- enlever des pages physiques au processus dont le taux est trop bas afin de les redistribuer au processus qui manque de pages physiques ;
- appliquer le « swapping out » de processus pour diminuer le degré de multiprogrammation, et libérer des pages physiques.

En fait le S.E. tente d'égaliser les taux de défaut de page pour tous les processus du système.