UNIVERSITE A.MIRA-BEJAIA
FACULTE DES SCIENCES EXACTES
DEPARTEMENT INFORMATIQUE

SUPPORT DE COURS
SYSTEMES D'EXPLOITATIONS
GESTION DE LA MEMOIRE

I. Introduction

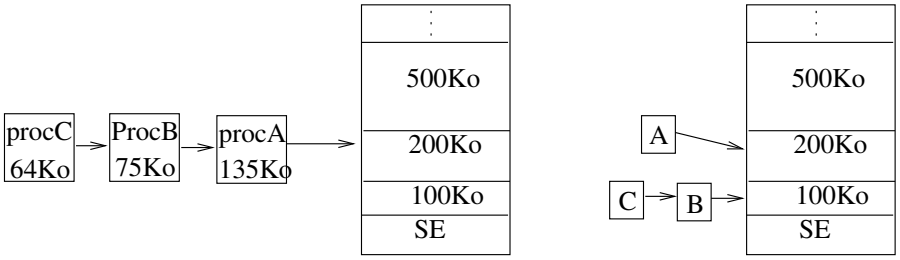
La gestion de la mémoire est du ressort du gestionnaire de mémoire. Il doit connaître les parties libres et occupées, allouer de l'espace mémoire aux processus qui en demandent récupérer de la mémoire libéré par des processus qui se terminent et gérer le va-et-vient (swapping).

Fonctionnalités du gestionnaire :
- Allocation : suivant des algorithmes d'allocation
- Mapping : translation d'adresses
- Réallocation : reloger des processus pour libérer une plus grande zone libre.
- Partage : gérer les zones mémoire partagés par plusieurs processus
- Protection : les adresses générées par un processus doivent être contrôlés à l'exécution pour s'assurer qu'elles ne font référence qu'a l'espace mémoire alloué à ce processus.
- Organisation logique : traditionnellement les adresses physiques vont de 0 à N et les programmes sont structurés en module.
- Organisation physique : La mémoire centrale (faite de semi-conducteurs) est de faible capacité mais rapide (temps d'accès ≈100ns),
La mémoire auxiliaire (magnétique ou optique) est de grande capacité mais lente (temps d'accès ≈100ms). Donc le gestionnaire doit gérer aussi le transfert entre les deux mémoires.

II. Gestion sans va-et-vient :

1. Mono-Programmation: Un seul processus présent en mémoire centrale (MC).
Il existe trois organisations possibles :
- Le Système d'exploitation (SE) en RAM et le processus utilisateur en RAM.
- Le SE en ROM et le processus utilisateur en RAM.
- Pilotes de périphériques en ROM (BIOS : Basic Input Output System), le SE en RAM et le processus utilisateur en RAM.
Deux registres sont nécessaires :
- registre de base : appelé B, contient l'adresse de la 1ere instruction du programme et les autres adresses sont calculées relativement à cette adresse.
- registre de séparation (limite) : appelé L, contient la dernière adresse possible alloué au processus qui sépare dons la partie utilisateur du système.
Le mapping : Soit 'a' une adresse quelconque appartenant à un programme :
i) f(a)=a+(B)
ii) si f(a)>(L) alors a est invalide (gestionnaire émet un message d'erreur).

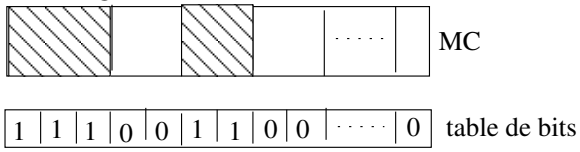
2. Multi-Programmation: plusieurs processus partagent la MC : cela évite la perte de temps de chargement et permet le pseudo-parallélisme. Le degré de multi-programmation est le nombre de processus possible pour une occupation maximale du processus.
Le mode de gestion dans ce cas est la gestion par partitions fixes (MFT : Multi-programming with a Fixed number of Tasks) : La MC est divisé en n parties de taille fixe (partition). Pour gérer ses partions on utilise soit :
-une file d'attente des processus (taches)
- plusieurs files d'attente : une par partition.



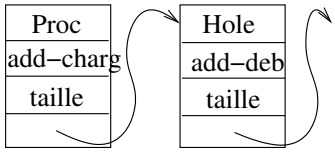
Chaque partition dispose d'un registre de base et de séparation. Donc le mapping est pareil que le précédent.

III. Gestion avec va-et-vient: Le mécanisme va-et-vient (swapping) consiste à stocker en mémoire auxiliaire (Disque Swap) les processus non-actifs et à les ramener en MC lors de leur activation.
Le Disque Swap (Swap Area) : est une partition du disque dur temporairement réservé pour recevoir les processus.
La gestion par partitions fixes n'est pas avantageuse car le processus qui laisse sa place à un autre n'est pas automatiquement de même taille. Cette gestion risque de générer une perte d'espace mémoire (fragmentation).
Donc le mode de gestion utilisé dans ce cas est la gestion par partitions variables (MVT : Multi-programming with a Variable number of Tasks) : La MC est dynamiquement divisée en un nombre variables de parties (partitions) de taille variable et de position variable.

Il existe trois méthodes de gestion par partition variable :
1. Gestion par table de bits: la MC est divisé en unités de taille fixe et à chaque unité on fait correspondre un bit dans une table : '0' pour unité libre et '1' pour occupé.
La taille de l'unité peut être de quelques mots (words) à plusieurs Kilo octets (Ko). Si la taille est trop petite on obtient une table de grande taille et vice-versa.



2. Gestion par liste chainée: il en existe deux sortes :
- une seule liste chainée d'unités de mémoire de taille variable. Chaque élément de la liste contient une indication si elle est libre ou pas, l'adresse de début de l'unité, sa taille et bien sûr un pointeur vers l'élément représentant l'unité suivante.



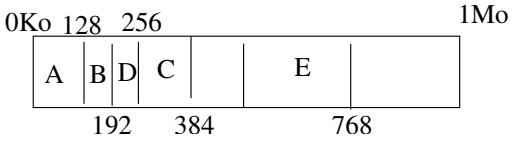
- deux listes chainées : une pour les zones occupées et une autre pour les zones libres.
Quand un processus se termine on marque la zone libre et on fusionne éventuellement avec des blocs adjacents.

Ce mode de gestion peut provoquer une fragmentation de la MC. Il existe deux sortes de fragmentation :
- Interne : l'espace mémoire vide (trou) appartient à l'espace mémoire d'un processus.
- Externe : l'espace mémoire vide n'appartient à aucun processus.

Le problème de fragmentation est résolu par un compactage (défragmentation) de la mémoire. Les processus sont relogés d'une manière compacte pour libérer une plus grande partition. La compaction (effectué par un processus système) nécessite l'arrêt de tous les autres processus et occupe du temps CPU.

3. Gestion par subdivision (Zones siamoises): (Buddy system)
La mémoire est dynamiquement divisés an parties de taille égale à une puissance de 2 (1 mot, …1Ko…1Mo) et les partitions sont gérées par des listes chainées : une liste par taille.
Initialement la MC est une seule partition et donc une seule liste.

Exemple : l'état d'une MC de 1Mo après les opérations de chargement suivantes : A(70Ko), B(35Ko), C(80Ko), D(60Ko), E(200Ko).



4. Algorithmes d'allocation de mémoire: il existe 5 méthodes possibles.
- Algorithme du meilleur ajustement (Best-fit) : On alloue au processus demandeur l'espace mémoire libre de taille la petite qui puisse le contenir.
- Algorithme du premier ajustement (First-fit) : On alloue la première zone libre trouvé qui puisse le contenir.
- Algorithme du pire ajustement (Worst-fit) : On alloue la zone de plus grande taille.
- Algorithme du prochain ajustement (Next-fit) : L'allocation suit l'algorithme du premier ajustement mais à partir de la dernière zone allouée.
- Algorithme de l'ajustement rapide (Quick-fit) : utilisé pour la gestion par subdivision. Il existe une liste de zones libre par taille.

La simulation de ces algorithmes a montré que le 'first-fit' est l'algorithme optimal. Le 'best-fit' produit une fragmentation importante et il est très coûteux en espace temps.

5. Récupération de la mémoire: le compactage (défragmentation) de la MC est très coûteux (en temps et matériel).
Certains systèmes permettent à des utilisateurs d'allouer de la mémoire à leur processus (malloc) et la libérer (free). Si on oublie de le faire le pointeur sur cette zone est perdu (la zone allouée devient inaccessible).
Le problème est plus important dans les systèmes orientés objet. Le Java (JVM) utilise la récupération automatique (Garbage Collector). Un thread de basse priorité arrête les autres threads et processus, balaie les objets en mémoire et dès qu'il trouve un objet sans référence valide, il le marque comme libre. A la fin de ce balayage il compacte la mémoire (les objets) utilisée. Ce thread se déclenche périodiquement.

IV. Mémoire Virtuelle:
L'idée est énoncée en 1961 par Fotheringham. C'est un mécanisme de translation d'adresse qui transforme des adresses fictives (virtuelles) dont dispose le programme en adresses physiques (réelles).
L'espace d'adressage virtuel de taille N est non-linéaire alors que l'espace d'adressage réel (MC) de taille m est linéaire avec $m \ll N$.
La gestion est effectuée par le MMU (Memory Managment Unit) qui permet la traduction d'une adresse virtuelle en adresse réelle suivant une application : $f : N \rightarrow m$.
L'utilisateur utilise une mémoire de taille N sans se préoccuper de la taille de sa MC (m).

La MV se compose de l'espace MC et se prolonge sur le DD (autre que la zone Swap).
Remarque : En général c'est un bit dans le PSW qui indique si le système utilise ou pas la MV.

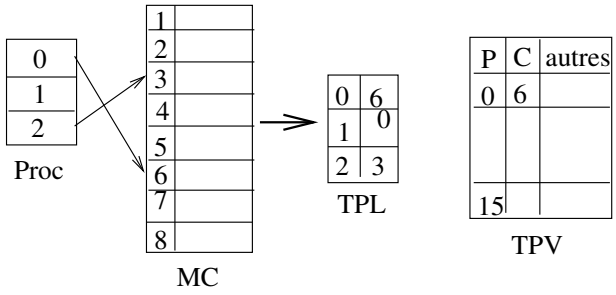
La MV peut être réalisé de trois manières : Les registres, la pagination ou la segmentation.

1. Registres: A chaque processus on associe deux registres : le registre de base (B) contient l'adresse de la première instruction du processus et sa réallocation est effectuée en modifiant uniquement le contenu de B. Le registre limite (L) contient l'adresse la plus haute alloué à ce processus. Le mapping reste le même comme précédemment.
Si un processus est interrompu, le contexte du processus est sauvegardé dans une pile y compris le contenu de B et L.

2. Pagination: On partage virtuellement l'espace d'adressage du processus en pages de taille fixe donc l'espace d'adressage du système (la MV) est donc partagé en pages de taille fixe. La MC (physique) est aussi partagé en parties (appelé cadres, cases, page-frames) de taille fixe et égale à la taille de la page.
Lors d'un chargement d'un processus sur la MC ses pages sont placées (chargées) dans des cadres vides de la MC. Un processus peut avoir certaines de ses pages uniquement chargées en MC (pages actives) et le reste (pages inactives) en MA (DD). Donc la MC contient les pages actives des différents processus.

Pour conserver une trace de l'emplacement des pages chaque processus dispose d'une table de pages et le gestionnaire aussi dispose d'une table globale de pages virtuelles (TPV).

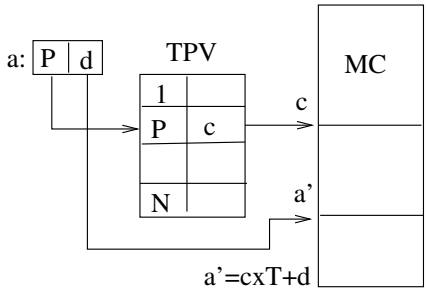
Exemple: soit une MC de 32Ko et une MV de 64 Ko, si une page fait 4Ko donc la MV se compose de 16 pages virtuelles (la TPV a 16 entrées) et la MC de 8 cadres.



Fonctionnement : A chaque fois qu'une page est nécessaire (lors d'un chargement d'un nouveau processus ou le calcul d'une adresse qui appartient à une nouvelle page) :
i) le gestionnaire détermine quelle page et son emplacement
ii) si la page n'existe pas en MC (en MA) => défaut de page.
iii) faire le chargement de la page si un cadre vide existe
iv) s'il n'existe pas de case vide (MC remplie) alors il faut choisir une des pages en MC à décharger (page victime).
v) si la page victime a été modifiée (en MC) alors elle doit être copiée (écrite) sur la MA.

Table des pages virtuelles : elle contient les numéros des pages virtuelles, leur N° da cadre (si elles sont chargées), un bit de présence, un bit de modification, un bit de référence etc.

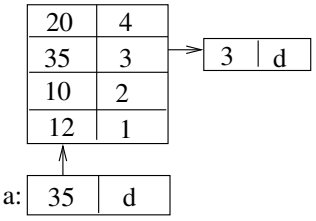
Une adresse virtuelle se compose d'un champ numéro de page et un champ déplacement.
Soit a=(P,d) une adresse virtuelle,



P : N° de page : indique une ligne dans la TPV.
C : N° de cadre : indique l'adresse de début du cadre (la taille étant connue).
d : le déplacement au niveau de la page (ou du cadre).
Le mapping : f(a)=C*Taille+d
Si C est nul (ou inexistant) alors c'est un défaut de page (donc la page est sur MA et il faut la charger sur la MC).
Exemple : si la MV a 2^{20} pages virtuelles quelle est la taille de la TPV ?

En général si la taille de la page est trop petite la taille de la TPV est trop grande à parcourir et a stocker en MC. Les solutions utilisées sont une mémoire associative juste pour la TPV ou une TPV à niveaux.

Mémoire Associative: c'est un ensemble de registres qui contiennent les adresses des pages actives (RAP) cela permet un gain de temps de parcours et d'espace mémoire. Nombre de RAP est égal au nombre de cadres !
Exemple : Une mémoire à 4 cadres :
a : adresse virtuelle, a' : adresse réelle : a'=3xtaille seg+d.



On peut aussi utiliser une mémoire associative pour les pages actives les plus récentes et une TPV normale.
Si le nombre de RAP est trop grand, on utilise une table de pages pour chaque processus et une mémoire associative pour les pages les plus demandés. Dans ce cas chaque RAP contient l'identité du processus à qui la page appartient.
Exemple : le DEC10 dispose d'un adressage virtuel de 512 pages de 512 mots donc une TPV de 256 mots.

TPV à niveaux: Une première table ayant comme entrée des indices vers d'autres tables de pages (le parcours est plus rapide). La première table est appelé la table des hyper-pages et en général il n'existe que deux niveaux.
Exemple : La VAX dispose d'un espace d'adressage de 4 Go et des pages de 512 octets. L'espace est partagé en 4 parties indexé par XY. Une adresse virtuelle (30 bits) est comme suit :

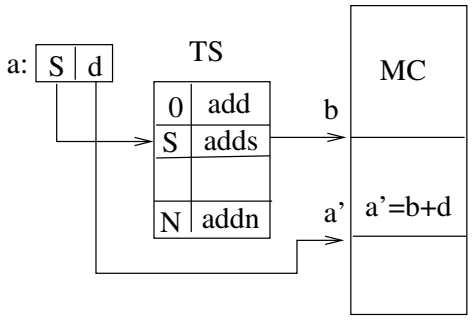
XY	NPV	dep

XY (2bits) : indique la partie
NPV (21 bits) : numéro de page virtuelle
Dep (9bits) : déplacement
Les valeurs de XY peuvent prendre les valeurs suivantes :
00 : espace utilisateur

01 : pile utilisateur
10 : SE
11 : réservé.

3. Segmentation: chaque processus possède un certain nombre segments de plusieurs Ko de taille variable. Les quatre premiers segments sont dédiés respectivement au code du programme, à la pile, aux données du programme et au TAS. Les autres segments peuvent être utilisés pour d'autres opérations par exemple contenir chacun un fichier, ou recevoir des données du réseau etc. (en général uniquement 4 segments !).
Le SE maintient pour chaque segment un descripteur de segment (comme le descripteur de fichier) qui contient :
- un identificateur de segment
- la taille du segment (en Ko ou en pages !)
- les droits d'accès (rwx !)
- dates de chargement, de modification, etc.
- d'autres paramètres en option (par exemple la durée en MC ou en MA)
- adresse de début de segment (adresse de chargement) ou un pointeur vers une table de pages.

Le PCB de chaque processus contient les descripteurs des segments de celui-ci.
Un segment de données peut être partagé par plusieurs processus avec différents modes d'accès.
Une adresse virtuelle a=(S,d)



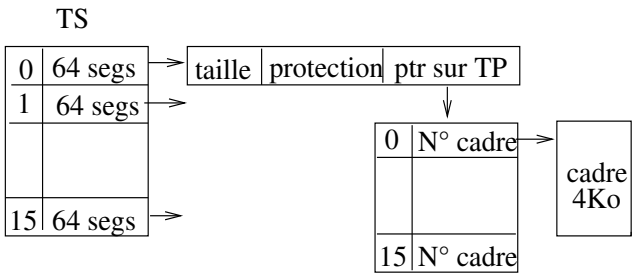
S : N° du segment
b : adresse de chargement du segment
d : déplacement au sein du segment chargé en MC.
Le mapping : f(a)=b+d
Si b est inexistant dons le segment n'est pas encore chargé alors il faut le charger
Si la MC est remplie alors il faut décharger un segment victime (mais le problème est plus complexe car les segments sont de taille variable !)
Les caractéristiques de la table de segments est la même que la TPV mais en général le nombre d'entrées de la TS est plus petit que le nombre d'entrées de la TPV.

4. Segmentation paginée: certains segments (code ou données) peuvent avoie de grande taille (donc la segmentation ou la pagination seule est inefficace). Donc on pagine les segments.
La MC est partagée an cadres et les programmes sont segmentés et chaque segment est partagé en pages de même taille que le cadre. On associe à chaque segment une table de page et il existe une table globale de segments (TGS).
Fonctionnement : une adresse virtuelle est un triplet a=(S,P,d)
i) le gestionnaire indexe la table TGS avec S
ii) si l'entrée TGS(S) est vide alors défaut de segment (le segment n'est pas chargé en MC).
iii) sinon l'entrée indique une table de pages (TPV)
iv) il indexe la TPV avec P
v) si l'entrée est vide défaut de page (la page n'est pas chargée en MC)
vi) sinon l'entrée indique un N° de cadre C

vii) l'adresse réelle est calculé : f(a)=C+d

Lors d'un défaut de page ou de segment, il faut charger le segment ou la page manquante dans la MC. Il existe des algorithmes d'allocation (pour les segments) pour trouver ou les placer et il existe aussi des algorithmes de remplacement de pages (ou de segments) pour trouver quelle page victime décharger.

Pour encore gagner du temps on utilise une mémoire associative comme table de segments et dans ce cas l'adresse à comparer est soit le triplet (S,P,d) ou le couple (S,P) pour connaître le numéro du cadre C.
Exemple: Le MMU des processeurs Motorola 68000 peut gérer au maximum 16 processus dans 64 segments. Chaque segment (représenté par un descripteur dans la TS) se compose de 16 pages au maximum. Chaque page fait 4Ko.



Donc un processus dispose de 4Mo et une adresse virtuelle est comme suit :

	Indice TS		Indice TP	
N°processus	N°segment		N°page	deplacement
4bits	6bits		4bits	12bits

V. Stratégies d'Allocation de la Mémoire : Il existe trois stratégies :
- Stratégie de chargement (Fetch) pour répondre a la question quand allouer ?
- Stratégie de placement : pour répondre à la question ou charger ?
- Stratégie de remplacement : pour répondre à la question quelle page éjecter (Quoi) ?

1. Stratégie de Chargement : Il existe le chargement à la demande ou chargement par anticipation (pré-chargement).

- A la demande : Une faute de page ou de segment génère une requête de chargement puis on utilise une stratégie de placement.
- Par anticipation : on peut prédire les futurs défauts de page (ou de segments) suivant deux paramètres :

➢ la nature de la construction des programmes : (le principe de la localisation) les références d'un programme tendent à être localisés en des points précis de l'espace mémoire et ces localisations ont tendance a ne changer que par intermittence.
➢ L'influence de la conduite du processus dans le passé: les références répétés sont enregistrés et dès le démarrage du processus les pages souvent référencés sont chargées automatiquement.

2. Stratégie de Placement : Surtout pour les systèmes segmentés ou il faut effectuer une insertion. Les segments sont insérés ou extraits et les zones libres (trous ou holes) ainsi formés sont gérés en liste chaînée par l'adresse du premier mot.
Si le segment à insérer est de taille plus petite que la taille de la zone libre alors il sera chargé à la fin de la zone pour ne pas avoir à modifier la liste chaînée.

Si la taille du segment est trop grande alors un compactage de la mémoire sera nécessaire.

* *Algorithmes d'insertion* :

i) le plus mauvais choix : les trous sont enchaînés dans l'ordre décroissant de leur taille . le segment sera inséré à la fin du premier grand trou et le nouveau trou ainsi formé sera inséré dans la liste.

ii) Le meilleur choix : les trous sont enchaînés dans l'ordre croissant de leur taille et la liste est parcourue à chaque chargement.

iii) Le premier trouvé : les trous sont enchaînés dans l'ordre croissant de leur adresse puis la tête de liste est avancée à chaque insertion

iv) Les frères siamois : les segments ont obligatoirement des taille puissance de 2 et pour chaque puissance une liste est crée triée dans l'ordre croissant de leur adresse.

3. Stratégie de Remplacement : Pour libérer de la place dans la mémoire il faut éjecter des pages ou des segments. Pour les systèmes paginés il faut éjecter autant de pages que nécessaire et pour les systèmes segmentés il faut éjecter autant de segments (adjacents !) jusqu'à avoir un trou correspondant à la taille du segment à charger.

* *Algorithmes de remplacement* : comment choisir la page ou le segment à éjecter ?

i) le moins récemment utilisé (LRU) : on suppose que le futur ressemblera au passé, donc on enregistre la dernière date de référence à une page ou un segment.

ii) le moins fréquemment utilisé (LFU): une page non référencée est non-utilisée, donc on comptabilise le nombre de références à une page.(compteur de références). En général les pages récentes ne sont pas concernées.

iii) le plus ancien (FIFO) : la page la plus vieille, donc on enregistre la date de chargement d'une page ou un segment (en général c'est aussi la pus utilisée !)

iv) Aléatoire (Random) : une page est choisie au hasard pour être éjectée

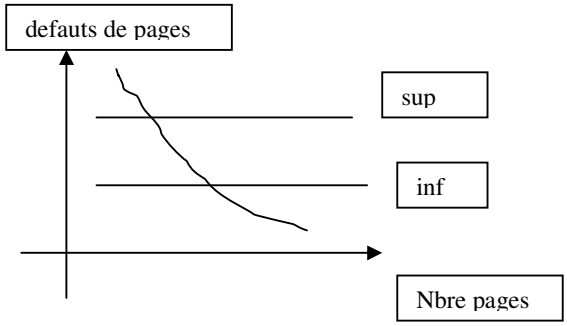
v) Algorithmes de l'horloge (FINUFO) : plusieurs paramètres sont combinés pour choisir une victime, donc la date de chargement et le compteur de références. Etc.

vi) Optimal : la page qui ne sera pas référencée pendant la durée la plus longue dans le futur sera remplacée.

VI. Considérations Techniques:

1. Ensemble de travail : (Working set) Chaque processus a besoin d'un nombre minimum de pages à maintenir en M.C. Si ce n'est pas le cas alors le processus est continuellement interrompu par des fautes de pages. Pour éviter cela on charge on charge ce qu'on appelle le working set du processus. Le working set est défini par l'ensemble des pages récemment référencés.

2. Ecroulement : (thrashing) c'est quand le nombre de défauts de pages dépasse une limite supérieure. En général on définit les limites inférieures et supérieures et à chaque fois qu'un processus n'excède pas la limite inférieure le système lui retire des pages et si le processus dépasse la limite supérieure on lui alloue des pages supplémentaires.



3. Politiques d'allocation :
- Allocation équitable : Le nombre de cadres par processus est fixe.
- Allocation proportionnelle : Le nombre de cadres est ~ à la taille du processus.
- Allocation locale : la page à supprimer doit appartenir au processus qui a provoqué le défaut de page.
- Allocation globale : la page à supprimer sera choisi parmi toutes les pages du système.

4. Taille des pages :
- Taille des pages trop grande : fragmentation interne, réduction de la TPV réduction du nombre de défauts de pages.
- Taille des pages trop petite : moins d'informations inutiles en M.C., TPV trop grande, perte de temps de chargement et déchargement des pages.
En général la taille idéale est de 724 octets, mais on utilise souvent des pages de 512 octets.

5. Verrouillage de la page : Si un processus est en attente de données par une opération d'entrées/sorties (ex : téléchargement) la page qui reçoit les données ne doit pas être déchargée donc on la verrouille !

6. Localité de référence :
- Chaîne de référence : la liste de numéros de pages à laquelle un processus accède durant sa vie.
- Localité temporelle : si une page a déjà été référencé alors il y'a de fortes chances qu'elle soit référencée de nouveau.
- Localité spatiale : si une page i a été référencée alors la page i+1 a de fortes chances d'être référencée.

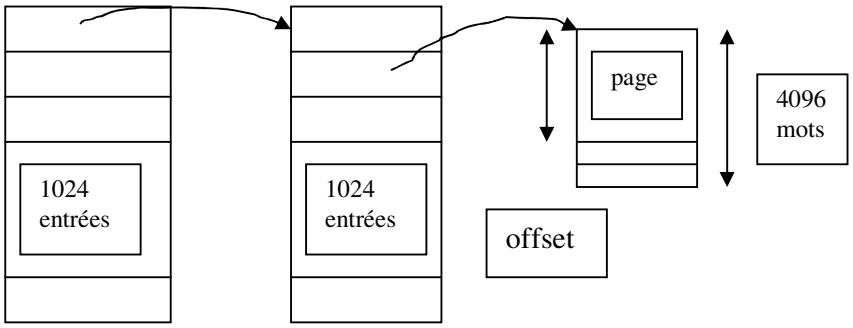
7. Performances des algorithmes de remplacement de pages :
Les algorithmes sont comparés en fixant le nombre de cadres et une chaîne de référence d'un processus donnée, ou pour un seul algorithme en variant le nombre de cardes mais en fixant la chaîne de références.
Anomalie de Belady : Plus de cadres n'implique pas toujours moins de défauts de pages.

8. Retour sur instruction : Une instruction est formée de plusieurs octets. Si l'adresse du premier opérande provoque le défaut de page donc une interruption du processus alors au retour de l'interruption le processus reprend à partir de l'instruction suivante.
La solution : un registre spécial est prévu pour stocker l'adresse de l'instruction précédente.

VII. Exemples :
1. Le microprocesseur Intel 80386 : Il combine des techniques de mémoire paginée et de techniques de segmentation.
Il dispose de 2^{14} segments indépendants ayant une capacité max de 1milliard de mots de 32 bits.
La mémoire virtuelle est implémentée sur deux tables :

- table locale des descripteurs (LDT) propre à chaque processus pour contenir les descripteurs des segments de code, de pile et de données.
- Table globale des descripteurs (GDT) pour tout le système et partagée par tous les processus. Elle contient les descripteurs de tous les segments de tous les processus et du système.

Le 386 dispose de 16 registres segments : pour accéder à un segment, il charge une adresse virtuelle dans un registre cette adresse pointe sur l'une des table. Chaque entrée dispose de l'adresse de base du segment l'adresse limite et d'autres champs.
Le 386 dispose d'une pagination optionnelle : pages de 4 Ko. La pagination se fait sur deux niveaux : une adresse virtuelle a 3 champs :
Dir (directory): indice sur un tableau de pointeurs de pages (10bits)
Page : indice sur un pointeur sur une page réelle (10bits)
Offset : déplacement à l'intérieur d'une page (12 bits)



2. Système UNIX : utilise la segmentation. Chaque processus dispose de :
-un segment code : non modifiable, et peut être partagé
-un segment de données : propre à chaque processus et modifiable. La partie des données initialisée est chargée en même temps que le segment code et sa taille ne varie pas. Pour la partie des données non initialisés la mémoire leur est allouée que lors de leur chargement (à a demande) donc sa taille varie.
-un segment pile : propre à chaque processus et modifiable, il est géré par le système. Au lancement du processus il contient les variables d'environnement et la ligne de commande du shell.
Le système UNIX utilise le swapping pour gérer le conflit d'utilisation de la M.C. entre les processus. On y a rajouté la pagination (à la demande) pour gérer les grands processus. Un démon (voleur de pages) vérifie régulièrement (toutes les 250 ms qu'un nombre fixe de cases vides soit disponible.

3. Système DOS : très proche du processeur Intel8086. Il utilise le partitionnement fixe et le swapping.
Le bus d'adresses a 16 bits⇒64Ko d'espace d'adressage. Les registres d'adresses ont 20 bits ⇒1Mo d'espace d'adressage. En fait la partie réservée aux utilisateurs est de 64Ko le reste est réservée aux pilotes des périphériques.
Grâce à une ligne d'adresse supplémentaire (ligne 21) on a ajouté 64Ko qui est exploitée par le système.
Maintenant les processeurs Intel permettent un adressage étendu : jusqu'à 16Mo (80286) et 4Go (≥80386).

16Mo	Mémoire étendue	Cache et
1Mo+64Ko	Mémoire haute	RAM
1Mo	Mémoire supérieure	E/S, ROM, Pilotes
64Ko	Mémoire conventionnelle	Prog. utilisateurs

4. Système Windows : Le mécanisme de fenêtrage nécessite de multiple allocation et libération de la mémoire, ce qui résulte par une fragmentation externe importante. Le système utilise donc beaucoup le compactage.
Il dispose d'une mémoire segmentée avec une table de segments à deux niveaux : une adresse virtuelle indique une entrée sur une première table qui elle-même désigne une autre entrée dans une seconde table et cette dernière désigne un segment.

Windows NT (New Technology) : Il dispose d'une mémoire paginée et la taille de chaque page est de 4 Ko. Son MMU se nomme VMM : Virtual memory unit.
Un défaut de page provoque le chargement de la page et une page de agerde. L'algorthme de remplacement de pages est FIFO.
Chaque processus a un repertoire de pages (1K d'entres) qui designe une table des pages (1K d'entrées) qui designe une page chargée.
⇒ adresse virtuelle :

Indice dans répertoire	Indice dans table	déplacement	
31	21	11	0

5. Système MACH : (système réparti)
Il gère la mémoire en objet : un objet mémoire peut être une page, un ensemble de pages ou un fichier ou toute autre structure de données spéciale.
A un processus on alloue une région qui est un ensemble d'objets mémoire. Les processus sont un ensemble de threads : les threads gère la mémoire alloué au processus père via des messages.
Un message contient la fonction désirée et il est adressé au port d'autres processus.
Exemple de fonctions de gestion de la mémoire :
-allocate/deallocate : alloue ou invalide une région d'un processus
-map : mapper un objet dans l'espace virtuel d'un processus qui retourne une adresse de base.
-inherit : positionne les attributs d'héritage d'une région.
-read/write : permet à un thread de lire/écrire une donnée d'un objet mémoire d'un autre processus.

-FIN-