Manuel 4
Gestion de la mémoire centrale
Plan du chapitre 4

1. Rôle architectural rempli par la mémoire centrale

1.1. Objets contenus en mémoire centrale

La mémoire centrale, ressource physique, est appelée à stocker :
- des objets passifs, comme des répertoires, des fichiers (de programmes, de textes, d'images, de sons), des tampons de données d'entrées-sorties ou de transfert réseau,
- des objets actifs, comme les processus.

Si les objets stockés ne comprennent pas de références internes sous forme d'adresse, ils ne posent pas trop de contraintes de placement ou de mobilité. Mais si ce sont des objets composés et si des adresses ont été utilisées pour désigner les composants de l'objet, ou si ce sont des processus, les adresses sont utilisées d'une manière spécifiques qui découle du schéma de Von Neumann.

Le *schéma d'un ordinateur* tel que l'a introduit *Von Neuman*, contient deux idées fondamentales.

De la première idée, celle du programme enregistré, il découle que toute instruction du processus, en particulier la prochaine à exécuter, doit être présente en mémoire centrale pour que le processus puisse aller la lire avant de l'exécuter.

De la deuxième idée, celle de la mémoire adressable, il résulte que l'unité centrale ne désigne pas une valeur mais l'adresse de la mémoire centrale où elle est rangée. Un processus doit faire un calcul d'adresse (indexation, adressage indirect, hachage, translation,...) qui lui donne accès à l'objet sur lequel porte l'opération à effectuer. Seuls les objets qu'il peut adresser lui sont accessibles.

La mémoire a donc un double rôle, le premier rôle est de repérer les objets par une adresse et seuls les objets qui ont une adresse sont accessibles, et le second rôle est de stocker les valeur des objets, sous forme codée. L'adresse d'un objet donne accès à son contenu. Les adresses du calcul d'adresse sont des objets d'un type particulier (pointeur, référence, index) et à ce titre leurs valeurs doivent aussi être stockées en mémoire centrale.

1.2. Rôle fondamental de la mémoire pour l'exécution d'un programme

1.2.1. *Inventorier les objets du processus (aspect mémoire virtuelle)*

Divers objets ("data") peuvent être nécessaires à l'exécution du programme, que ce soient les instructions, pointeurs et données du programme, les bibliothèques qu'il peut appeler

statiquement ou dynamiquement, directement ou par fermeture transitive, l'environnement que le système met en place pour ce programme, les éléments du système qu'il peut appeler sous contrôle des appels systèmes, directement ou par fermeture transitive, les objets externes atteignables par des opérations d'entrées-sorties. Tous ces objets doivent être énumérables par le processus. Cette énumération doit comprendre aussi bien les objets connus au début du processus que les objets créés ou acquis dynamiquement pendant l'exécution. Le processus doit pouvoir différencier chaque objet (et chaque composant élémentaire d'objet, en général l'octet) pour pouvoir le distinguer des autres quand le calcul en a besoin (quand il y fait référence). Il faut classer et énumérer tous les objets en leur donnant des noms différents, ce qui se traduit, en mémoire centrale, par des adresses différentes. Pour faire cette énumération et ce classement, on n'a pas à connaître la valeur de l'objet (mais on doit connaître son encombrement).

L'énumération et le classement des objets d'un processus est préparé tout au long de la chaîne de production de programme (compilation, édition de liens, chargement, lancement). Quand on passe de l'énumération par nom symbolique (identificateur alphanumérique dans les programmes sources, dans les catalogues de fichier ou de nommage) à un classement par adresse en mémoire, il faut tenir compte de la taille des objets en mémoire centrale, car chaque objet est exprimé comme un multiple d'octet. La taille du processus est la taille (en général en nombre d'octets) de l'espace d'indexation (ou d'adressage) nécessaire pour classer tous les objets en mémoire centrale (et ensuite pour les y ranger, car on ne peut ranger qu'un objet qui a été classé).

Le mécanisme d'indexation qui permet d'associer un numéro unique (appelé adresse virtuelle) à chaque objet du processus est appelé l'espace d'adressage virtuel du processus (ou encore mémoire virtuelle). On considère que tout objet est décomposable en octets et en général on réserve un numéro pour tous les octets de l'objet (cela revient à tenir compte de la taille de l'objet) . En général, on utilise un espace d'adressage linéaire (on dit aussi que c'est un espace plat, "one level flat address space") avec un index appelé adresse (virtuelle). Quelquefois on utilise un ensemble d'espaces disjoints, chacun étant linéaire. C'est l'adressage segmenté avec deux index, un numéro de segment et une adresse dans le segment. Les segments sont réellement disjoints et le successeur de la dernière adresse d'un segment n'est pas la première adresse du segment de numéro suivant.

A l'exécution du programme, c'est la fonction calcul d'adresse réalisée dans l'unité centrale lors du décodage de l'instruction courante qui fournit l'adresse effective (virtuelle) de l'objet concerné par l'exécution de l'instruction courante du processus et dont la valeur est à ranger ou à rechercher en mémoire. Une unité centrale ne désigne jamais une valeur, mais donne son index de classement dans l'espace d'adressage virtuel.

L'espace d'adressage virtuel et les conventions de son utilisation doivent être connus et respectés par tous les intervenants qui préparent ou exécutent le processus :

- l'architecture du processeur : adresses d'interruption, de récupération des appels systèmes, ...
- le système d'exploitation : frontière mode maître - mode utilisateur, protections, gestion mémoire,...
- la chaîne de production de programme : zone de code, zone de constantes, zone de données statiques, zone de données dynamiques, zone de la pile d'exécution.

A ce stade, on sait énumérer et repérer tous les objets du processus, mais on ne sait pas où ils sont réellement ni quelle est leur valeur. C'est grâce à cette virtualisation (désignation virtuelle) qu'on peut préparer des programmes indépendamment de l'ordinateur, et exécuter un même programme avec des données différentes.

1.2.2. *Conserver et stocker les valeurs des objets inventoriés*

Selon le schéma de fonctionnement d'un ordinateur, la valeur d'un objet manipulé pendant l'exécution d'un programme doit, en dernier ressort, être stockée en mémoire centrale physique pour pouvoir être relue ou modifiée par l'unité centrale. Ce lieu de stockage est divisé en contenants de la taille du composant élémentaire, en général l'octet (en fait des groupes d'octets consécutifs). Ce lieu de stockage est géré en utilisant les adresses de contenants physiques. La valeur d'un objet est fournie à l'unité centrale en allant lire le contenu d'un contenant dont on a l'adresse de stockage physique. De même l'unité centrale range la valeur d'un objet numéroté (ils le sont tous) dans un contenant d'adresse de stockage donné.

Certaines valeurs des objets d'un processus (instructions, constantes, variables initialisées) sont préparées par la chaîne de production de programme (compilation, édition de liens, chargement, lancement). Les valeurs codées sont conservées dans des fichiers et servent à élaborer l'image exécutable initiale du processus. Cette image est stockée en mémoire secondaire (en Unix c'est un fichier d'un répertoire /bin) et est chargée totalement ou partiellement en mémoire centrale avant le lancement du processus.

D'autres valeurs sont élaborées par le processus pendant son exécution et sont donc stockées en premier lieu en mémoire centrale. On rappelle que, même pour une valeur inconnue au démarrage du processus, il faut avoir réservé une adresse virtuelle dans l'espace d'adressage du processus ou savoir en obtenir une dynamiquement (dans la plile ou dans le tas).

1.3. Placement des objets classés pour un processus

On a vu qu'on effectuait le classement des objets avant l'exécution des processus et indépendamment d'elle, mais qu'à l'exécution les valeurs de ces objets devaient avoir une adresse de stockage en mémoire centrale. Il faut établir à un moment une relation entre classement et stockage et passer de la mémoire virtuelle (adresses désignées dans les instructions et les données du processus) à un stockage physique dans la mémoire centrale et dans la hiérarchie des mémoires matérielles. Le moment où on fixe cette relation détermine la manière de réaliser le placement en mémoire centrale des objets d'un processus

1.3.1. *Stockage et classement confondus dès la conception*

Si le classement se fait par le biais du stockage en mémoire, comme c'était le cas des premiers ordinateurs, la relation est fixée dès la conception du programme. L'adresse virtuelle et l'adresse physique sont confondues. Cela introduit alors trois rigidités :
- les valeurs doivent être rangées selon des adresses contiguës en mémoire physique (séquentialité des adresses d'instructions successives, sauf instruction de saut d'adresse)
- les valeurs des objets d'un processus ne peuvent pas être déplacées sur leur support sans modifier aussi le classement de ces objets. Le seul déplacement acceptable est de tout déplacer en bloc, à condition d'avoir un registre de translation..
- l'espace d'adressage ne peut pas être plus grand que la mémoire centrale. Tous les objets d'un processus doivent tenir en mémoire centrale. On triche en faisant du recouvrement ("overlay") dans l'espace d'adressage virtuel.

1.3.2. *Indépendance entre stockage et classement*

Pour obtenir l'indépendance entre stockage et classement, on introduit une fonction de topographie qui place (ou mappe) l'espace d'adressage virtuel dans l'espace de stockage en mémoire centrale (dans le cas de la pagination, ce mappage est défini avec l'aide d'une unité de topologie cablée, la MMU ou “Memory Management Unit”). Cette fonction peut être utilisée pendant l'exécution du processus à la fin de chaque calcul d'adresse. On établit aussi un placement de l'espace d'adressage virtuel dans un espace de stockage en mémoire secondaire ; celui-ci contient l'image exécutable initiale du processus et éventuellement l'image courante quand tous les objets ne peuvent être stockés en mémoire centrale (disque de va et vient).

1.3.3. *Monoprogrammation et multiprogrammation*

Le système fonctionne en monoprogrammation quand à tout instant un seul processus utilisateur est placé en mémoire centrale. Il est géré en multiprogrammation quand plusieurs processus se partagent l'unité centrale et la mémoire centrale. La multiprogrammation a été introduite d'abord pour augmenter le taux d'utilisation de l'unité centrale quand les programmes ont un taux élevé d'entrées sorties, puis pour gérer le partage de l'unité centrale dans les systèmes interactifs en temps partagé.

Lorsque plusieurs processus cohabitent dans la mémoire centrale, la gestion de mémoire doit assurer la protection mutuelle d’informations appartenant à des processus différents tout en permettant le partage d’informations entre eux lorsque nécessaire (isolation des mémoires virtuelles et mappage vers des informations partagées)

1.3.4. *Nature et moment du placement*

Le placement peut intervenir uniquement au démarrage du processus et être complet. Dans ce cas la localisation du processus en mémoire centrale est unique et définitive. l'implantation du processus en mémoire centrale est statique.

Si on utilise un mécanisme de migration de l'image du processus (va et vient), un processus bloqué (ou prêt) doit libérer la mémoire qu'il occupe et, à chaque nouvelle activation (respectivement élection) du processus, on peut vouloir soit retrouver la même place soit refaire un placement complet. Dans ce dernier cas, la localisation est provisoire et l'implantation en mémoire centrale est dynamique.

On peut encore se limiter à un placement partiel de quelques objets au démarrage du processus et les objets complémentaires (segments ou pages) sont placés à la demande du processus en exécution, au fur et à mesure qu'il en découvre le besoin. Il faut alors un mécanisme cablé pour détecter, après le calcul d'adresse par l'unité centrale, l'absence (ou défaut) de segment ou de page

1.3.5. *Politique de placement par zones.*

Dans la politique de placement par zones, l'image exécutable est stockée en un seul bloc d'adresses contiguës en mémoire centrale. Il faut alors translater toutes les adresses de l'image pour la placer dans le bloc libre. Cette translation se fait une fois pour toute ou à chaque retour en mémoire centrale si l'image du processus subit un va-et-vient avec la mémoire secondaire ("swapping"). La translation se fait dynamiquement avec un registre de base qui permet d'ajouter une valeur fixe (celle de l'adresse de la base du bloc alloué) à toute adresse calculée, avant son envoi par l'unité centrale sur le bus d'adresses mémoire. Un registre limite permet de vérifier que l'adresse reste dans le bloc alloué.

1.3.6. *Politique de placement par pagination.*

Pour mettre en oeuvre la politique de placement par pagination, l'espace d'adressage et l'image sont découpés par morceaux de taille fixe appelés pages. Chaque page est stockée dans une case de mémoire centrale. Les cases ne sont pas nécessairement contiguës, mais une transformation topographique des adresses permet de conserver la contiguïté logique des adresses virtuelles du processus. Cette transformation se fait dynamiquement avec une mappe mémoire ("MMU ou memory management unit"), mécanisme câblé qui permet, pour toute adresse calculée par l'unité centrale, de remplacer l'adresse de page par l'adresse de case, avant son envoi sur le bus d'adresses mémoire.

L'image peut être placée complètement (mais par pages non contiguës) en mémoire centrale. Mais elle peut aussi n'y être placée que partiellement, ce qui permet d'avoir des images beaucoup plus grandes que la mémoire physique installée. Il faut alors que l'unité centrale puisse détecter qu'une page n'est pas chargée dans une case de mémoire centrale, et que ce défaut de page alerte le système pour qu'il charge la page manquante. C'est le mécanisme de pagination à la demande avec va et vient. Le temps d'exécution d'un processus en pagination à la demande varie avec le nombre de défauts de pages. La politique de gestion de la mémoire centrale doit en limiter le nombre.

1.4. Gestion de la mémoire secondaire

La mémoire secondaire doit stocker l'image exécutable initiale, puis l'image exécutable quand elle est sauvegardée par va et vient ("swapping"), et enfin l'image finale résultat ("dump").

On utilise une table de localisation en mémoire secondaire pour repérer le placement de l'image. Le placement en mémoire secondaire peut être statique (emplacements fixes) ou dynamique.

Lorsqu'on utilise la pagination, la mémoire secondaire est en général découpée en granules de même taille que les pages et le placement peut y être fait par granules contigus ou non. (la contiguïté permet de réduire la durée des accès disques, mais elle complique la gestion quand le processus augmente la taille de son image initiale).

S'il n'y a qu'un niveau de mémoire secondaire (et donc pas de va et vient avec un autre niveau), l'allocation dynamique de granules de mémoire secondaire contient un risque d'interblocage qui augmente quand on approche du remplissage total de cette mémoire.

2. Placement par blocs contigus

2.1. Placement des processus

Toute l'image du processus (code, donnée, pile, tas) est en mémoire centrale. Toute la mémoire virtuelle d'un processus est donc placée dans la mémoire centrale. Celle-ci contient le système d'exploitation et un ou plusieurs processus. Il faut éviter qu'un processus n'aille faire un accès dans le système ou dans l'image d'un autre processus ; c'est le rôle des registres de base et de limite, complétés quelquefois par des verrous d'accès sur les blocs de mémoire physique. Le mot d'état du processeur doit contenir la bonne clé pour que l'accès soit autorisé.

En monoprogrammation, on ne traite qu'un programme à la fois, il n'y a donc qu'un seul processus, et l'unité centrale reste inactive pendant les entrées sorties. Les autres programmes attendent dans une file et les politiques de service peuvent être très variées. Si l'image du processus est plus grande que la mémoire disponible, on doit programmer du recouvrement d'adresse virtuelle ("overlay") entre certaines sections qui n'ont pas à être ensemble en mémoire.

En multiprogrammation fixe (appelée aussi technique MFT), la mémoire est prédécoupée en partitions de tailles fixes, différentes entre elles pour tenir compte des tailles des programmes. On a une file d'attente par partition ou bien une file unique. Un processus occupe une partition et s'il y a va et vient, il peut revenir dans une partition différente. Si l'image d'un processus augmente de taille pendant l'exécution, il faut déplacer le processus pour le mettre dans une partition plus grande. La translation d'adresse qu'entraîne le changement de partition se fait en modifiant la valeur du registre de base.

En multiprogrammation variable (technique MVT), le nombre de partitions dépend de la taille des processus qui remplissent la mémoire disponible. Avant de placer un processus (au démarrage ou après un va et vient), il faut trouver une zone de taille suffisante pour le contenir en entier. Si l'image d'un processus augmente de taille pendant l'exécution, il faut trouver une zone plus grande pour y déplacer le processus, ou l'envoyer en mémoire secondaire en attendant qu'une telle zone existe. On peut aussi agrandir la taille allouée après avoir envoyé en mémoire secondaire le processus qui occupe le bloc voisin. On alloue des blocs libres et comme leur taille peut être plus grande que la taille demandée, il reste alors un bloc libre de plus petite taille, qu'on appelle un résidu. On assiste ainsi à une fragmentation des tailles des zones libres, même si on essaie de regrouper en un seul deux blocs libres adjacents. On peut avoir une situation où le total des tailles libres est supérieur à la taille demandée, mais où aucun bloc ne l'est individuellement. On a appelé ce phénomène la fragmentation externe. Dans cette situation, on ne peut faire le placement contigu demandé sans auparavant recompacter la mémoire.

2.2. Allocation par zones

Il y a plusieurs méthodes pour allouer une zone libre de taille Z, $Z \geq T$, où T est la taille du processus à placer. Par exemple, on prend la première zone libre qui convient, c'est le premier accord ("first fit"), ou bien on recherche la zone qui laisse le plus petit résidu Z - T, c'est le meilleur accord, ou encore pour ne pas avoir trop de petites zones, on recherche la zone qui laisse le plus grand résidu Z - T, c'est l'accord le plus ample ("worst fit"). Des simulations ont été faites dès 1965 [Shore, 65] pour connaître la meilleure utilisation de la mémoire et elles ont montré que le premier accord et le meilleur accord donnaient des résultats comparables et que l'accord le plus ample était bien moins bon. La représentation des zones libres se fait par chaînage des zones libres ; on utilise souvent le premier mot de la zone pour contenir l'adresse de la zone suivante et le deuxième mot pour contenir la longueur de la zone. L'allocation d'une zone libre selon le meilleur accord est simplifié si le chaînage est fait selon les tailles croissantes. A la libération d'une zone, il faut regarder si la zone libérée n'est pas voisine d'une zone déjà libre, et si c'est le cas les regrouper en une zone libre unique et de plus grande taille. Cette opération est simplifiée si les zones libres sont chaînées par adresses croissantes. En général, c'est le chaînage retenu pour la méthode du premier accord.

Quelle que soit la méthode d'allocation par zone, lorsque le nombre N de zones allouées est grand et que le système est en équilibre (c'est à dire que le nombre moyen des zones libres est constant), le nombre de zones libres M est approximativement égal à N/2. Cette règle dite des 50% [Knuth, 68] s'obtient en observant que l'évolution de M dépend de N et de la répartition géométrique des N zones allouées.

Une zone occupée peut être adjacente de zéro, une ou deux zones libres. Quand A, une zone occupée entourée de deux zones libres, est libérée, on la regroupe avec les deux libres adjacentes pour ne faire plus qu'une zone, M diminue de 1. Quand B, une zone occupée contiguë à une seule zone libre, est libérée, on la regroupe avec la zone libre adjacente pour ne faire plus qu'une zone, M est inchangé. Quand C, une zone occupée entourée d'aucune zone libre, est libérée, M augmente de 1. S'il y a nA zones de type A, nB de type B, nC de type C, la libération des N zones allouées entraîne NC augmentations de M et nA diminutions de M.

N et M peuvent être exprimées avec ces valeurs :

(1) N = nA + nB + nC -- nombre de zones allouées

(2) M = (2*nA + nB)/2 + e -- nombre de zones libres

avec e = 0, 1 ou 2 selon la configuration des bords de la mémoire. On néglige e << M

D'autre part les allocations font aussi varier M le nombre de zones libres. Si p est la probabilité qu'une allocation laisse un résidu (p est voisin de 1), alors M ne change pas avec la probabilité p et M diminue de 1 avec la probabilité 1 - p. Donc les N allocations entraînent N*(1 - p) diminutions de M.

A l'équilibre, par définition, le bilan est nul en moyenne temporelle, donc :

"l'augmentation moyenne de M compense la diminution moyenne de M"

soit :

nC = nA + N*(1 - p), ou encore : nC - nA = N*(1 - p)

Or en prenant une combinaison linéaire des équations (1) et (2), il vient, en négligeant e, :

N - 2*M = nC - nA , d'où

N - 2*M = N*(1 - p) donc 2*M = N* p CQFD

En dehors de la mémoire centrale destinée à contenir l'image d'un processus, objet actif, cette méthode d'allocation par zones contiguës est utilisée pour des objets passifs créés par les programmes, les serveurs d'objets, de fichiers ou de base de données (textes, images, sons), les caches web, les gestionnaires de messages,... Les demandes sont imprévisibles, les tailles demandées sont très variables. Si on sait que le nombre de tailles est limité, on peut avoir une file de blocs libres par taille et dans certains cas utiliser des méthodes spécifiques qui facilitent le regroupement des blocs libres ("buddy system", "Fibonacci").

3. Placement par pagination

3.1. Principe et mise en oeuvre de la pagination

3.1.1. *Principe de la pagination.*

L'espace d'adressage (mémoire virtuelle) est divisé en pages de taille fixe. La mémoire centrale est découpée en cases de même taille. Chaque page est rangée dans une case de mémoire centrale. Les cases ne sont pas nécessairement contiguës.

Pour passer de l'espace d'adressage paginé à la mémoire centrale, lieu de stockage pour exécution, on met en place une transformation (ou mappage) topographique qui associe une page de l'espace d'adressage virtuel à une case de même taille dans la mémoire physique. Ce mappage intervient à chaque référence à la mémoire virtuelle, donc plusieurs fois par instruction machine. Il est indispensable qu'il soit cablé.

3.1.2. *Mise en oeuvre de la pagination*

La transformation topographique est donc réalisée par câblage avant de lancer l'ordre d'accès à la mémoire centrale. On découpe l'adresse virtuelle en deux parties : une partie déplacement dont la taille maximale est égale à la taille de la page et une partie numéro de page qui est transformée en numéro de case. Et c'est l'adresse obtenue par concaténation de numéro de case et du déplacement qui est envoyée sur le bus d'accès à la mémoire centrale. C'est l'adresse physique où l'information désignée par l'adresse virtuelle est (ou sera) stockée.

Le mappage est mémorisé dans une table des pages qui indique si la page virtuelle PVi fait partie de l'espace d'adressage du processus Pi, quel est le type d'accès autorisé dans cette page (aucun, lecture seule, écriture, exécution) et quelle est la case de mémoire centrale qui lui est associée et quel est le numéro Cf de cette case. Ce numéro Cf est utilisé pour effectuer la transformation d'adresse.

Chaque entrée i de la table des pages PV sert à gérer une page virtuelle PVi et elle contient :

- I : booléen de présence, indiquant l'association avec une page physique. Il sert aussi de témoin de défaut de page (et d'invalidité de la zone Cf).
- V : verrou d'accès à la page virtuelle (aucun accès, lecture seule, exécution, écriture et lecture). C'est le moyen de gérer les accès à la mémoire virtuelle.
- E : témoin d'écriture dans la page depuis la réinitialisation du témoin. Cela permet de savoir que l'image sur disque n'est plus cohérente avec la page en mémoire centrale.
- parfois des témoins d'utilisation de la page, permettant de repérer si elle a été référencée depuis la réinitialisation du témoin.
- Cf : case de mémoire associée quand le mappage est présent (quand I est vrai)

Il y a nécessairement une table des pages par processus. Cette table des pages est stockée en mémoire. Dans les premiers systèmes paginés, elle était de petite taille et elle pouvait être copiée entièrement dans une mémoire rapide et utilisée directement par le mécanisme câblé.

Quand l'espace d'adressage est grand, on utilise un accélérateur constitué par une mémoire associative (appelée aussi "translation look-aside buffer" ou TLB), conservée dans l'unité centrale. On y stocke les derniers couples (PVi, Cf) utilisés et cette table est consultée d'abord par le mécanisme câblé. Si PVi n'y figure pas, on fait la recherche dans la table des pages de Pi. Quand PVi fait partie de la mémoire virtuelle de Pi, le mappage se trouve inscrit dans la table des pages et le mécanisme l'utilise et l'inscrit dans la mémoire associative où il remplace le couple le plus ancien (en fait on peut utiliser les techniques de remplacement de page qu'on verra plus loin avec la pagination à la demande). La performance de cette mémoire associative dépend de la probabilité d'y trouver un couple (PVi, Cf) et elle repose sur la propriété de localité des programmes et sur la taille de la mémoire (en général de 64 à 2048 entrées). À l'élection d'un nouveau processus, les couples stockés pour le processus précédent n'ont plus de sens et il faut tous les invalider.

L'utilisation de la mémoire associative illustre de façon spectaculaire la propriété de localité des programmes. Par exemple, une mémoire associative de 64 entrées peut résoudre avec succès environ 99% des références faites par 16 processus avec chacun un espace d'adressage de 2^{20} pages. La mémoire associative ne mémorise pourtant qu'une page sur 256 000.

Quand l'espace d'adressage est très grand (64 bits d'adressage), la table des pages, qui devient très grande (2^{20} entrées soit 10^6 entrées), ne peut plus tenir entièrement en mémoire centrale et elle est elle-même paginée (c'est toujours le cas, mais c'est moins gênant quand l'espace d'adressage n'est pas trop grand). On utilise alors deux techniques pour limiter le temps d'accès. Dans la méthode de la table des pages inverse, on crée une table des case de la mémoire centrale et chaque case Cf y possède une entrée qui permet de préciser le processus Pi qui l'utilise et la page virtuelle PVi mappée. La recherche d'une page virtuelle PVi d'un processus Pi se fait par la technique de hachage (avec un éventuel chaînage des entrées qui correspondent au même résultat de la fonction de hachage). Dans la méthode du mappage hiérarchique, on découpe l'espace d'adressage en pages et superpages et on utilise une table des superpages pour regrouper des adresses de tables de pages. Cette méthode utilise la propriété de localité au niveau de la table des pages : quand les références successives d'un programme font accès à deux pages, la probabilité est plus grande qu'elles soient dans une même superpage que dans deux superpages différentes. Cette méthode s'applique parfois à plus de 2 niveaux hiérarchiques. Ces grands espaces d'adressage impliquent la pagination à la demande. Cette technique sera vue plus loin.

3.1.3. *Considérations complémentaires*

Par construction, il n'y a plus de fragmentation externe, toutefois il existe une fragmentation interne qui traduit le fait que certaines pages ne sont pas pleines. Cette fragmentation est limitée car elle peut être contrôlée par le compilateur, l'éditeur de liens ou tout progiciel qui place des références d'objets dans l'espace d'adressage du processus.

La protection repose sur le contrôle d'accès qui s'exerce sur chaque page et qui utilise des indicateurs d'accès (aucun accès, lecture seule, lecture et écriture, exécution). Les tentatives d'accès illégales sont déroutées vers le noyau du système. D'autres indicateurs servent à gérer le

va et vient des informations (et le remplacement pour la pagination à la demande) : témoin de mappage avec une case, témoin de modification (“dirty page”), témoin de référence.

Le partage de pages de code réentrant ou de pages de données entre des processus complique la gestion de la pagination et introduit une concurrence d'accès aux tables des pages. On introduit parfois des tables des pages spéciales pour les objets partagés quand des conventions d'adressage virtuel régissent l'utilisation de ces objets partagés. On a alors des architectures avec plusieurs tables de pages par processus : une table des pages non partagées du processus, une table des pages pour la bibliothèque partagée, une table des pages pour le code système (partagé) qui s’exécute en mode utilisateur.

La taille des pages est un élément important car les programmes contiennent simultanément des petits objets (descripteurs de ressources et de processus) et de gros objets (fichiers de texte, d'image). Le choix résulte d'un compromis entre fragmentation interne et taille de la table des pages. Certaines architectures essaient d'introduire plusieurs tailles de pages. Cela entraîne une gestion beaucoup plus complexe.

Les entrées sorties tamponnées, par canal ou accès direct à la mémoire, ne doivent pas être interrompues. Donc, ou bien on verrouille en mémoire centrale les cases d'un processus pendant le transfert, ou bien on interdit toute entrée sortie dans l'espace virtuel de l'utilisateur. Dans ce cas les données doivent être copiées dans un espace système, introduisant un surcoût important.

La localité des programme est une hypothèse souvent utilisée pour gérer les processus paginés. L'augmenter en restructurant les programmes au niveau du compilateur ou de l'éditeur de liens est très bénéfique.

3.2. Placement total par pagination

Dans ce type de placement, toute l'image du processus (code, donnée, pile, tas) doit être placée en mémoire centrale. Celle-ci contient le système d'exploitation et un ou plusieurs processus. Quand un processus de N pages doit être chargé pour exécution, il faut lui trouver N cases (non nécessairement contiguës) pour stocker toute l'image du processus. Ce type de placement est très utilisé dans les applications temps réel, où la variabilité des accès à l'information, et partant des temps de réponse, entrainée par la pagination à la demande, ne peut être tolérée.

4. Autres utilisations dynamiques de la mémoire

Dans les architectures à adressage segmenté, l’espace d’adressage est composé de segments indépendants. Chaque segment peut être soit associé à une zone de mémoire contiguë, soit paginé. On dit qu’on a une mémoire virtuelle segmentée.

En dehors du stockage des processus en exécution, la mémoire est utilisée et allouée dynamiquement pour des stockages intermédiaires ou persistants de données. C’est le cas de serveurs en mémoire centrale ou sur disque pour :
- des systèmes de fichiers centralisés ou répartis,
- des bases de données,
- des tampons de messages dans les réseaux,
- des cache web,
- des serveurs de flux multimédia (image video, son,..),
- des serveurs de coopération pair à pair,
- des serveurs de mobilité.

On retrouve dans ces serveurs les techniques d’allocation par zone ou par pagination, ainsi que les politiques de remplacement qui ont été vues ici.

On notera que les techniques de remplacement sont aussi présentes dans les caches employés comme accélérateurs divers. C’est le cas de la mémoire associative (TLB) utilisée pour la pagination.