CHAPITRE 5

ALLOCATION À LA DEMANDE ET STRATÉGIES GLOBALES D'ALLOCATION

Plan

Manuel 5

Allocation à la demande et Stratégies globales d'allocation

1. Pagination à la demande

1.1. Placement partiel

Dans un système qui fonctionne en pagination à la demande, seule une partie du processus est résident en mémoire centrale, parce que l'image du processus est plus grande que la mémoire centrale qui lui est allouée. C'est le cas quand la taille de la mémoire centrale totale est de plus petite taille que l'espace d'adressage ou quand on veut partager cette mémoire centrale entre de nombreux processus et n'en attribuer qu'une portion à chacun. La table des pages indique alors si une case est associée ou non à une page donnée. Quand une case n'est pas associée à la page référencée, le mécanisme cablé déclenche un défaut de page. C'est l'exécution du processus qui détermine les pages qui ne sont pas placées en mémoire physique (on dit que ce sont les pages "manquantes") et les pages manquantes peuvent être chargées à la demande, au fur et à mesure des occurrences de défaut de page. Avant l'élection du processus, on peut faire un préchargement des pages qu'il est le plus probable que le processus utilisera en premier (quand on les connaît ou quand on sait en faire une estimation correcte).

Mais dans ce cas le problème change de nature. Quand on n'ordonnance l'unité centrale (ou les unités centrales) qu'entre les processus dont on a placé toute l'image en mémoire

physique, que ce soit par zone ou par pagination, on est dans le cas où les ressources, mémoire physique et unité(s) centrale(s), sont bien allouées l'une après l'autre. Quand on fait de la pagination à la demande, on mélange les deux allocations et on ne peut plus se contenter de gérer chaque ressource isolément. Il faut une politique globale pour contrôler la coopération entre les deux types de demandes de ressources. Ne pas avoir pris en compte cet aspect a conduit au phénomène d'écroulement de système. Avant d'examiner la nature de la politique globale à mettre en place, on étudie le mécanisme à mettre en oeuvre pour traiter le défaut de page lors de pagination à la demande.

1.2. Traitement du défaut de page

Si toute l'image du processus ne peut être placée en mémoire, il faut en garder une copie en mémoire secondaire et faire évoluer cette copie en même temps que le processus. La table des pages contient maintenant, pour chaque page virtuelle de l'espace d'adressage, une adresse de case de la mémoire centrale quand le mappage existe (comme il n'existe pas toujours, le mappage est partiel) et une adresse de granule de la mémoire secondaire (granule de la taille d'une page). Il y a toujours un granule qui est associé à chaque page de l'espace d'adressage (À tout le moins, pour toute page initialisée, quand on alloue dynamiquement un granule lors de la première écriture dans une page).

Lorsque, pour le processus Pi, le calcul d'adresse (l'adresse obtenue après indexation ou indirection avec des données contenues dans les registres internes est parfois appelée adresse effective) détermine une référence à la page virtuelle PVi et qu'il n'y a pas de case de mémoire centrale qui lui est associée dans la table des pages, le mécanisme de pagination génère un déroutement (ou interruption) pour défaut de page. L'exécution du processus est interrompu et le système est appelé. Le traitement du défaut de page comprend d'abord l'analyse de la cause de l'appel et la vérification que la page manquante PVi fait bien partie de l'image du processus.

Si PVi n'est pas dans l'espace d'adressage du processus, c'est une erreur de programmation et le déroutement engendre une exception ou simplement détruit le processus.

Si PVi fait bien partie de l'espace d'adressage, il faut lui mapper une case Cf. Soit il existe une case libre et dans ce cas, il faut y copier le contenu du granule de mémoire secondaire qui est associé à PVi. Comme ce transfert depuis le disque est long (pendant ce temps on peut exécuter environ 10 000 instructions) le processus Pi est bloqué et l'unité centrale est allouée à un autre processus prêt. S'il n'y a pas de case libre, il faut au préalable en libérer une, ce qui peut entraîner la sauvegarde de son contenu vers la mémoire secondaire avant de pouvoir utiliser la case pour PVi. Quand la case Cf est chargée, le système reprend la main pour corriger la table des pages de Pi et noter l'existence du mappage entre PVi et Cf. Puis il met Pi dans l'état prêt. Quand Pi est élu à nouveau, l'instruction qui a causé le défaut de page est recommencée et la transformation d'adresse pour PVi se passe bien si le mappage existe encore (c'est à dire si Cf n'a pas été reprise entre temps pour un autre processus). Notons que si une instruction fait référence à plusieurs adresses, chacune d'elle peut entraîner un défaut de page.

1.3. Coût du défaut de page

Le temps d'accès à la mémoire centrale, qui est de 10 à 100 nanosecondes sans défaut de page, peut atteindre 10 millisecondes avec défaut de page. La probabilité de défauts de page doit rester faible. La probabilité de défauts de page p et le temps d'accès relatif S entre mémoire centrale et mémoire secondaire sont les deux paramètres principaux quand on modélise l'efficacité de l'utilisation de la pagination à la demande. Cette efficacité e(p) se met sous la forme : $e(p) = 1 / (1 + p*S)$.

Les taux de défauts de page ont été mesurés en faisant varier la taille de mémoire centrale allouées à un processus, c'est à dire le pourcentage de pages résidentes. Le taux de défauts de page reste faible pour une grande taille d'allocation puis il apparaît une rapide élévation de ce taux en deçà d'un certain seuil.

1.4. Remplacement de page

La pagination à la demande entraîne le remplacement de page quand il faut trouver une case pour une nouvelle page et qu'il n'y a plus de case libre. La politique de choix de la page à remplacer peut varier : cette victime peut être choisie localement parmi les pages du processus

demandeur, ou globalement parmi toutes les pages résidentes ou encore par un mélange des deux. Le remplacement local maintient un nombre fixe de pages résidentes par processus et un nombre fixe de processus ; on est en multiprogrammation fixe. Le remplacement global permet une multiprogrammation variable. On verra plus loin que cela peut conduire à l'écroulement du système. La politique mixte vise à attribuer à chaque processus un nombre optimal de pages résidentes, nombre qui diffère d'un processus à un autre et qui évolue au cours du temps.

Le choix de la victime relève d'un algorithme de remplacement. Le meilleur choix (OPTIMUM) serait de remplacer une page qui ne servira plus ou la page résidente dont la prochaine utilisation est le plus loin dans le futur. Ce choix suppose la connaissance du futur du processus et n'est pas possible. On a essayé de choisir la victime au hasard ("RANDOM"), à l'ancienneté de résidence en mémoire centrale ("FIFO", first in first out), à l'ancienneté de référence ("LRU", least recently used). Ce dernier choix, qui est le meilleur quand on ne connaît pas le futur, s'appuie sur la localité des références et l'observation que les pages qui viennent d'être référencées ont une forte probabilité d'être à nouveau référencées.

Le choix de la victime peut tenir compte du fait que la page a été modifiée (et dans ce cas il faut d'abord sauvegarder son contenu dans la mémoire secondaire) ou non. Parfois , quand l'architecture cablée le permet, on tient compte du fait que la page a été accédée récemment.

L'effet des algorithmes de remplacement à été mesuré et comparé en faisant varier la taille de mémoire centrale allouée à un processus, c'est à dire le taux de pages résidentes. On a constaté que l'influence du remplacement était faible à coté du taux de pages résidentes. Cela a permis de relativiser le choix de l'algorithme de remplacement. Ainsi l'algorithme à l'ancienneté de référence ("LRU"), qui est trop coûteux à mettre en oeuvre, est-il remplacé par des approximations comme l'algorithme à l'ancienneté ou comme l'algorithme de la seconde chance. Dans ce dernier, on utilise le témoin de référence qui par câblage est remis à 1 à chaque référence à la page. Il est utilisé pour éviter qu'une page anciennement résidente, mais encore récemment utilisée, ne soit choisie pour victime. Les pages résidentes sont chaînées circulairement (à l'ancienneté) et une référence appelée l'horloge repère la première victime (donc en principe la page la plus ancienne). Si le témoin de référence de cette victime potentielle est à 1, on passe à la page suivante du chaînage jusqu'à ce qu'on trouve une victime dont le témoin de référence est à zéro. Pour éviter de boucler indéfiniment, le témoin de référence de toute victime non retenue est remis à zéro. L'horloge progresse aussi et désigne toujours la prochaine victime potentielle. Cet algorithme est aussi appelé algorithme de l'horloge. On peut raffiner le choix de la victime en utilisant aussi le témoin de modification qui, lorsqu'il existe, indique qu'une page résidente à été modifiée. Dans ce cas, on fait plusieurs parcours de la chaîne des pages pour y rechercher une victime d'abord dans les pages ni référencées ni modifiées, puis dans les pages non référencées, mais modifiées, ensuite dans les pages référencées et non modifiées et enfin dans les pages référencées et modifiées.

Le classement des algorithmes selon leurs performances moyennes est : OPTIMUM, LRU, Seconde Chance, Horloge, FIFO, RANDOM.

1.5. Politique d'allocation de pages

Le choix de base est de faire on non de la pagination à la demande. Si on a assez de mémoire centrale et si on ne peut accepter la trop grande variabilité temporelle due à la pagination (cas du temps réel), on place toute l'image du processus en mémoire centrale. C'est simple à gérer, la pagination évite l'allocation contiguë et la fragmentation externe.

Avec la pagination à la demande, se pose la répartition des cases entre les processus. Une allocation fixe peut être égalitaire en allouant le même nombre à chaque processus, proportionnelle à la taille du processus. Une allocation fixe entraîne un remplacement local où la victime est une page du processus demandeur. Le remplacement global, où la victime peut être choisie en dehors du processus demandeur, entraîne une variation de l'allocation. Quand la victime peut être choisie dans n'importe quel processus, on a une allocation sauvage qui peut conduire à l'écroulement. Cela conduit à contrôler l'allocation, en gardant un stock de victimes, hors allocation. Pour gagner du temps, un processus cyclique, démon de pagination, vide périodiquement les pages écrites qui sont dans le stock des victimes. Les processus prêts se divisent en deux classes, ceux qui sont allocataires et qui peuvent être élus, et ceux qui sont ajournés et dont les pages alimentent le stock de victimes. Avec un tel schéma, un processus qui, après avoir été ajourné, redevient allocataire et élu, peut faire un défaut de page pour une page qui

est encore dans le stock des victimes (cette page y a été placée pendant que le processus était ajourné). Le traitement du défaut de page doit, avant tout choix d'une victime, regarder si la page manquante est encore dans le stock des victimes et si c'est le cas il doit la récuperer.. Le stock des victimes joue un rôle de cache vis à vis de la mémoire de va et vient.

2. Écroulement du système

On a vu que pour améliorer le rendement du processeur quand le taux d'entrées sorties croissait, il fallait augmenter la multiprogrammation. On a vu également qu'en augmentant le partage de la mémoire, grâce à la pagination à la demande et au remplacement, on améliorait le partage du système entre les utilisateurs interactifs. La mise en oeuvre de ces deux objectifs indépendamment l'un de l'autre peut conduire à l'opposé du but visé. Au delà d'un certain seuil, le temps de réponse augmente brutalement, les performances sont fortement dégradées et on assiste à l'écroulement du système.

En analysant plus finement, on observe le phénomène suivant. L'allocateur détecte un faible taux d'utilisation de l'unité centrale et, pour améliorer le rendement, introduit un processus de plus dans l'état prêt. À son élection, ce processus, qui n'a aucune page résidente, fait un défaut de page, et le remplacement choisit une victime dans un autre processus prêt. Ceci augmente le trafic de va et vient et l'allocateur, qui constate que le taux d'utilisation de l'unité centrale reste faible, introduit à nouveau un processus de plus. Le taux de multiprogrammation augmentant, le partage de la mémoire augmente aussi, surtout s'il est égalitaire. Chaque processus prêt dispose de moins de pages résidentes et la probabilité de défaut de page augmente, ce qui accroît le trafic de va et vient, baisse le taux d'utilisation de l'unité centrale. Ceci incite l'allocateur à augmenter le taux de multiprogrammation et ce cercle vicieux conduit à l'écroulement.

Le schéma de fonctionnement d'un ordinateur selon Von Neumann nécessite l'utilisation conjointe de l'unité centrale et de la mémoire avec un ensemble de travail qui correspond à un minimum d'objets accessibles (code et données). Si, pour chacun des processus prêts, cet ensemble de travail devient insuffisant, le taux de défaut de page augmente, ce qui entraîne la baisse de la durée d'exécution entre deux défauts de page et l'augmentation du nombre de transferts de page. Puis augmente aussi la probabilité que le remplacement retire une page à un processus lui-même en attente de page ou au processus qui va être élu et qui référencera la page qui vient de lui être retirée.

3. Régulation de charge, mesures en ligne et stratégies globales

3.1. Stratégies pour éviter l'écroulement

Pour empêcher l'écroulement, il faut introduire une régulation de charge (ou contrôle de charge) qui mette en oeuvre une stratégie globale pour le choix du taux de multiprogrammation. Cette régulation dynamique s'appuie sur la mesure en ligne de paramètres de contrôle, comme le taux d'activité du processeur et le trafic de migration.

Le choix du bon taux de multiprogrammation doit résulter d'un compromis entre d'une part le nombre de pages résidentes, ce qui influence le nombre moyen de références faites à la mémoire centrale entre 2 défauts de page, et d'autre part les vitesses relatives du processeur et de l'accès au disque de va et vient.

Plus fondamentalement, il faut attribuer à chaque processus un ensemble de travail minimum avant de lui allouer l'unité centrale. On doit faire une allocation globale de mémoire et d'unité centrale.

L'utilisateur peut aussi avoir une action favorable en retravaillant son programme pour diminuer la taille de l'ensemble de travail. On verra que ce n'est pas négligeable.

3.2. États d'un processus avec une régulation de charge

La régulation divisent les processus en deux classes, ceux qui sont choisis pour participer à l'allocation globale, qui sont appelés allocataires et qui peuvent être élus, et ceux qui sont ajournés, qui ne peuvent recevoir l'unité centrale et dont les cases qui leur avaient été allouées peuvent être réquisitionnées. Les processus ajournés sont considérés comme évacués sur disque, même si cette évacuation peut être retardée pour n'être faite qu'en cas de nécessité

pour le bon fonctionnement des allocataires. Le changement d'état d'un processus entre ajourné et allocataire est déterminé par la régulation de charge.

Donc un processus qui pour l'utilisateur comporte deux états intrinsèques, actif ou bloqué, passe avec cette régulation par plusieurs états technologiques. Un processus actif, s'il est allocataire, peut être prêt ou élu. S'il est ajourné, il est gelé. Un processus bloqué, s'il est allocataire, est suspendu. S'il est ajourné, il est oisif

3.3. Premier type de régulation : PFF

Dans la régulation PFF ("page fault frequency") la charge est estimée par l'observation du trafic global de pagination et le contrôle se fonde sur les analyses de comportement de processus en mémoire restreinte. On a noté qu'en dessous d'un taux de pages résidentes le taux de défauts de page augmentait rapidement. Cela contribue donc à augmenter le trafic global de pagination. Le contrôle consiste à interdire cette augmentation et à supposer qu'en diminuant le taux de multiprogrammation, on contribue à diminue la fréquence des défauts de page des processus. Cette approche globale, qui se fonde sur un seul paramètre, a l'avantage de la simplicité et d'empêcher l'écroulement. Le choix d'un processus allocataire à ajourner, celui d'un ajourné à ajouter aux allocataires, l'attention à ne pas ajourner indéfiniment un processus, sont des fonctions qui doivent être traitées dans toute régulation. Un exemple en est donné dans la régulation par l'ensemble de travail.

3.4. Deuxième type de régulation : IBM 370 VM

Cette régulation est intéressante car elle a donné lieu à des expérimentations détaillées. La charge est estimée par deux paramètres qui sont l'activité de l'unité centrale et le nombre de pages remplacées pendant un intervalle Δt (Shils 1968 avec $\Delta t = 10$ s). Des seuils Suc et Spr pour ces paramètres déterminent des charges pour le système. Au dessus de Suc, l'activité de l'unité centrale est optimale et le système est en charge normale. Au dessous de Suc, on fait intervenir le trafic ; soit il est faible, en dessous de Spr, et le système est en sous-charge, soit il est fort, au dessus de Spr, et le système est en surcharge. Tous les Δt le régulateur intervient : en cas de surcharge, il exclut un allocataire ; en cas de sous-charge, il admet un allocataire de plus ; en cas de charge normale, il ne fait rien. Cette régulation évite l'écroulement et son gain a été mesuré en lançant de manière concurrente plusieurs versions du même programme. Au delà de 3 versions concurrentes sans régulateur, le système s'écroule. La présence du régulateur permet d'avoir une durée moyenne d'exécution qui reste a peu près constante.

Les changements d'états ont aussi été mesurés. On retrouve que l'on évite la surcharge, mais on détecte une forte oscillation entre la sous-charge et les autres états. La régulation de charge, comme toute méthode de régulation par asservissement, peut introduire une instabilité si il n'y a pas de phénomène d'amortissement ou d'hystérésis (ici ce serait de mettre des seuils différents selon le sens du changement d'état, par exemple de décréter la fin de sous-charge avec Spn + h et de ne quitter la surcharge que pour Spn - h).

3.5. Régulation par estimation de l'ensemble de travail

On utilise l'analyse fine des références d'un programme en mémoire virtuelle.

Les fondements de cette régulation reposent sur l'observation détaillée de l'utilisation de l'espace d'adressage par un programme séquentiel. La meilleure expression en est la représentation faite par Hatfield [Hatfield 72] qui donne une trace, pour une suite d'intervalle de 500 ms, des pages référencées pendant chaque intervalle. On y constate que pendant un intervalle un petit nombre de pages cumule la plupart des références ; il est courant de constater que 75% des références s'adressent à 20% des pages ; la répartition des références est non uniforme. On note aussi que d'un intervalle au suivant, la répartition des références paraît stable ; il est courant de constater que 90% des références d'un intervalle se retrouvent dans l'intervalle suivant ; l'observation des références d'un intervalle est une bonne estimation des références de l'intervalle suivant ; la répartition des références est stationnaire et on dit que les programmes possèdent la propriété de localité. Enfin le comportement du processus montre une succession de phases très stables et relativement prévisibles où la répartition des références est stationnaire ; ces phases, assez longues, sont séparées par des transitions courtes avec un comportement erratique marqué par une forte dispersion des références. Ce comportement peut s'expliquer par

l'importance des boucles de calcul et des structures de tableau et de pile qui concentrent les références pendant des phases de calcul séparées par des transitions correspondant à des branchements ou à des appels de sous-programmes.

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 taille de la mémoire associative ne lui permet de ne mémoriser pourtant qu'une page sur 256 000 seulement!

Ce comportement a été modélisé pour la pagination avec la notion d'ensemble de travail ("working set"). A tout instant t, l'ensemble de travail de fenêtre Δ, noté ET(t, Δ), est l'ensemble des pages qui ont été référencées au moins une fois entre les instants t - Δ et t. La taille de l'ensemble de travail est TET(t, Δ). La non uniformité des références entraîne que la taille de l'ensemble de travail est nettement inférieure à la taille du programme. La localité des références entraîne que l'ensemble de travail, et encore plus sa taille, restent stables sur plusieurs intervalles.

Pendant les phases de stabilité, les plus fréquentes, ET(t, Δ) est un bon estimateur de ET(t + Δ, Δ), et TET(t, Δ) est un très bon estimateur de TET(t + Δ, Δ). Les mesures ont montré que 90% des références dans ET(t, Δ) se retrouvaient dans ET(t + Δ, Δ).

La régulation qui utilise l'ensemble de travail s'appuie sur cette estimation et sur la règle suivante. Tout processus Pi, allocataire à t, doit pouvoir utiliser un nombre de cases égal à la taille de son ensemble de travail TETi(t, Δ). Ce critère définit le taux de multiprogrammation N à tout instant. La mémoire centrale est partagée entre le système résident, le stock de pages victimables, si l'allocateur en utilise, et les processus allocataires.

La taille du stock des pages victimables peut être un paramètre de la régulation.

Tout allocateur doit gérer les éléments de la régulation lors des événements significatifs.

Donnons quelques indications de cette gestion dans le cadre de la présente régulation.

Les défauts de page permettent de calculer la taille de l'ensemble de travail et d'en repérer les variations dans le temps. On calcule la taille de l'ensemble de travail courant en réinitialisant ce calcul à chaque élection du processus ou à chaque quantum.

Lors du traitement de chaque défaut de page, on incrémente la taille de l'ensemble de travail courant. et on essaie d'abord de récupérer la page manquante si elle se trouve encore dans le stock des pages victimables. Si la page manquante n'est plus en mémoire centrale, on applique une stratégie de remplacement. Soit un exemple de stratégie de remplacement mixte : si la taille de l'ensemble de travail courant est inférieure à la taille du précédent ensemble de travail, on prend une victime dans le stock des pages victimables, car on s'est garanti une taille de ce stock suffisante pour servir au minimum tous les ensembles de travail courants des processus allocataires. Si la taille de l'ensemble de travail courant du processus dépasse celle de son ensemble de travail précédent, le remplacement se fait dans le stock des pages victimables si possible, sinon dans l'ensemble de travail du processus demandeur.

En fin de quantum d'allocation de l'unité centrale ou lors de toute transition d'état depuis l'état élu, on arrête la taille de l'ensemble de travail du processus. Si on sait les identifier, les pages résidentes non conservées dans la nouvelle estimation de l'ensemble de travail sont placées dans le stock des pages victimables. Si la taille de l'ensemble de travail augmente, il faut vérifier que le critère qui définit le taux de multiprogrammation reste vrai, sinon il faut ajourner un des processus allocataires.

Pour éviter que les processus ajournés attendent trop longtemps, on reconsidère les choix d'allocation, soit périodiquement à un multiple de quantum, soit au départ et à l'arrivée d'un processus. Par exemple, on recalcule les priorités de tous les processus prêts, en tenant compte, dans ce calcul, du type de processus, du temps d'attente dans l'état ajourné, du temps passé dans l'état allocataire,... Puis on choisit les processus les plus prioritaires jusqu'à ce que le critère qui définit le taux de multiprogrammation ne s'applique plus. Les pages résidentes des anciens processus allocataires sont placées dans le stock des pages victimables.

Un processus, démon cyclique de pagination, est chargé de recopier sur disque les pages écrites, pour gagner du temps quand elle seront choisies comme victime. On ne doit pas les effacer car elles peuvent aussi être récupérées par leur processus propriétaire ou par un autre processus si elles contiennent de l'information partagée.

3.6. Actions de l'utilisateur intelligent

L'utilisateur peut aussi avoir une action favorable en retravaillant son programme pour diminuer la taille de l'ensemble de travail.

La structuration des programmes a une influence sur leur localité. Soit par exemple des pages de 2048 octets, soit 512 mots, et un tableau A : array(1.. 512, 1.. 512) of Integer; stocké ligne par ligne. Donc A est stocké :
A(1, 1), A(1, 2),...,A(1, 256), A(2, 1), A(2, 2),.... A(256, 256).

Avec l'itération :
for j in 1..512 loop for i in 1.. 512 loop A(i, j) := 0; end loop; end loop;
le parcours de l'espace d'adressage est AI(1, 1), A(2, 1),...Le processus lit un mot par page et passe à la page suivante. Avec une taille mémoire inférieure à 512 cases pour A (même si c'est 511 cases), il y aura 512 * 512 défauts de page.

Avec l'itération :
for i in 1..512 loop for j in 1..512 loop A(i, j) := 0; end loop; end loop;
le parcours de l'espace d'adressage est A(1, 1), A(1, 2), ...Le processus lit 512 mots par page avant de passer à la page suivante. Avec une taille mémoire de 2 cases (une pour le code, une pour le tableau A), il y aura seulement 512 défauts de page, soit 512 fois moins. Cela permet de réduire considérablement le temps d'exécution du processus (jusqu'à un facteur de 2).

La permutation de l'ordre de placement des modules en mémoire virtuelle pour regrouper dans un ensemble de pages minimal des modules qui s'appellent les uns les autres permet aussi de réduire la taille de l'ensemble de travail. On a constaté une diminution de 50% du taux de défaut de page.

4. Autres exemples de politiques globales

La nécessité d'une politique globale d'allocation se rencontre partout où il faut éviter les trop longues listes d'attente, qui sont la cause ou le témoin d'inéfficacité. On découpe cette allocation en étapes : les clients attendent dans une antichambre avant d'être servis au salon. On établit une stratégie globale qui régit les étapes et il y a une politique à chaque étape. Considérons quelques exemples.

Soit l'exemple de la prévention de l'interblocage lorsque des processus demandent des ressources par étapes successives et accumulent les ressources qu'ils ont déjà reçues. La prévention peut amener à bloquer tous les processus sauf un. S'il y a beaucoup de processus dans le système, c'est très pénalisant. Par prévention, on bloque des processus qui immobilisent des ressources déjà allouées parce qu'ils risquent d'en prendre encore plus et de placer le système en interblocage. Pour limiter le nombre de processus à mettre en attente alors qu'ils possèdent déjà des ressources, seuls quelques processus, en nombre limité, sont autorisés à accumuler des ressources , les autres attendent leur tour dans une antichambre. Cette hiérarchisation procure une moindre immobilisation de ressources dans un but de prévention et un meilleur temps de réponse moyen.

Pour ordonnancer des travaux en traitement par lots, on utilise une hiérarchie d'ordonnanceurs. L'ordonnanceur à long terme fait le choix des travaux à multiprogrammer (tient compte des taux d'E/S, des priorités, alloue les périphériques lents ou les disques amovibles à installer), l'ordonnanceur à court terme alloue le processeur aux processus prêts.

Une organisation hiérarchique permet de réagir avec tolérance aux fautes temporelles des applications temps réel : un régisseur élimine les travaux les moins importants jusqu'à ce que les travaux les plus importants puissent être ordonnançables, tandis qu'un ordonnanceur temps réel alloue dynamiquement le processeur aux travaux retenus par le régisseur.

5. Autres utilisations de la pagination ou du remplacement

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

En dehors du stockage des processus en exécution, la mémoire paginée 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 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.

Un emploi important du remplacement est fait dans les caches Web. En effet le développement de l'internet entraîne l'utilisation de caches Web (« proxy caches ») et un très grand nombre de stratégies de remplacement d'objets ont été implantées. On en trouve une récente synthèse dans la revue « ACM Computing Surveys », volume 38 , n° 4, de décembre 2003, pages 374 à 398 : « A survey of Web Cache Replacement Strategies », par S. Podlipnig et L. Böszörmenyi.