INF2610

Module 7 : Gestion de la mémoire

7.1 Pour chacune des adresses suivantes, 5867, 12675, 19283 et 162345, dites quel est le numéro de page et le décalage, pour des pages de taille 4Kio?

Il faut séparer le champ numéro de page (bits les plus significatifs) et le champ décalage (log2(taille de page) bits les moins significatifs, soit 12 bits pour 4Kio). Ceci peut se faire en convertissant en binaire et en extrayant les champs voulus. Il est parfois plus simple pour l'humain de prendre la partie entière de la division par la taille de page (numéro de page) ainsi que le reste de cette division (décalage). Ceci donne donc 5867/4096 donne 1 comme numéro de page et 5867 % 4096 donne 1771 comme décalage. Nous avons de la même manière (3, 407), (4, 2899) et (39, 2601) pour les trois autres chiffres.

7.2 Un ordinateur possède des adresses virtuelles de 32 bits et des pages de 8Kio. La table de pages est en matériel et doit être chargée à raison de 100ns par entrée, chacune faisant 32 bits. Quel est le temps requis lors d'un changement de contexte? Est-ce un problème si les changements de contexte arrivent aux 100ms?

La table de pages contient 4Gio / 8Kio = 524288 entrées. A 100ns par entrée, cela prendra 52ms, soit plus de la moitié de chaque quantum alloué à un processus. Si plusieurs entrées sont identiques (e.g. entrées à 0 au centre de la table) entre deux processus, il serait possible de le mémoriser et d'en tirer partie, et ainsi sauver beaucoup de temps.

7.3 Un ordinateur utilise une table de pages à trois niveau. Les adresses virtuelles sont donc décomposées en 4 champs (a, b, c, d), a contenant les bits les plus significatifs. Que détermine la taille de chacun de ces champs? Il y a souvent une relation précise entre la taille relative de ces champs sur la plupart des systèmes, expliquez?

Le champs d détermine la taille des pages (2^d). Les champs a, b et c déterminent le nombre d'entrées dans les sections de tables de niveau 1, 2 et 3 respectivement. Usuellement, une page est utilisée pour chaque section de table, quel que soit le niveau. Le nombre d'entrée dans la section est donc la taille d'une page divisée par la taille d'une entrée. De ce fait, a, b et c sont généralement égaux. La taille d'une entrée est usuellement la taille de l'adresse virtuelle (32 bits pour un système 32 bits et 64 bits pour un système 64 bits); ceci permet d'y stocker le numéro de page physique ainsi que quelques bits pour les permissions, le statut...

7.4 Un ordinateur possède des adresses virtuelles de 32 bits et des pages de 4Kio. Un programme n'occupe que la première page (pour son code et ses variables) et

la dernière page de la table (pour sa pile). Quel est l'espace requis pour la table de pages si le système utilise une table à un seul niveau? A deux niveaux?

Une table à un niveau requiert 4Gio / 4Kio = 1Mi entrées soit 4Mio. Dans une table à 2 niveaux, une page est utilisée au premier niveau avec deux entrées utilisées et deux pages au second niveau avec une entrée utilisée chacune. Il faut donc 3 pages pour la table de pages soit 12Kio.

7.5 Un ordinateur utilise une table de pages avec 1024 entrées pour chaque processus. La lecture dans la table de pages prend 5ns. Une cache de pré-traduction d'adresse (TLB) contient 32 entrées et peut être accédée en 1ns (au lieu de 5ns). Quel est le taux de succès requis afin d'avoir un temps d'accès moyen de 2ns?

Le temps moyen sera pour un taux de succès s : s * 1ns + (1 – s) * 5ns = 2ns. En conséquence, s = ¾ = 75%.

7.6 Un ordinateur avec une mémoire centrale de 256Mio et des pages de 8Kio possède un espace virtuel de 64 Gio et utilise une table de pages inversée. Quelle taille devrait avoir la table de hachage associée, qui doit être une puissance de 2, si on veut avoir un nombre d'entrée moyen par case inférieur à 1.

L'ordinateur possède 256Mi/8Ki = 32768 pages physiques. La table de hachage pourrait avoir le double, 65536 entrées, si on veut avoir un nombre moyen strictement inférieur à 1.

7.7 Un étudiant propose de construire un compilateur capable de produire une liste des pages qui seront accédées en séquence par un programme, afin d'implémenter l'argorithme optimal de remplacement de pages. Est-ce possible? Que peut-on faire pour aider l'algorithme de remplacement de pages?

Puisqu'on ne peut pas prédire même seulement si un programme peut se terminer ou non, il est encore moins possible pour le cas général de prédire à la compilation le chemin d'exécution dans le programme. Par contre, il est possible d'utiliser certaines informations à la compilation (e.g. arbre d'appel) pour optimiser la suite des opérations. En effet, les fonctions qui s'appellent l'une l'autre peuvent être placées sur une même page. On peut aussi donner des indices sur les pages qui risquent d'être requises peu de temps après une certaine page. Il est aussi intéressant de récolter de l'information à l'exécution. Par exemple, la séquence d'amorçage de l'ordinateur est toujours la même. Certains systèmes notent les pages lues une première fois et utilisent ensuite cette information pour demander une pré-lecture de ces pages en parallèle avec le début du démarrage.

7.8 Un petit ordinateur contient 4 pages physiques que se disputent 8 pages en mémoire virtuelle. Pendant l'exécution, la séquence d'accès des pages virtuelles est la suivante : 0172327103. Combien de fautes de pages surviennent avec la stratégie FIFO? LRU?

En premier, les pages 0172 sont chargées en mémoire. Ensuite, 3 remplace 0, puis 2, 7 et 1 sont là, il faut recharger 0 qui remplace 1, et 3 est là. Le total est 6 fautes. Avec LRU, les pages 0172 sont chargées en mémoire. Ensuite, 3 remplace 0, puis 2, 7 et 1 sont là, il faut recharger 0 qui remplace 3 (inutilisé depuis plus longtemps de 2, 7 et 1). Il faut finalement à nouveau lire 3 qui remplace 2. Le total est 7 fautes.

7.9 Un ordinateur fournit à chaque processus un espace de 65536 octets divisé en pages de 4Kio. Un programme utilise un espace de *text* de 32768 octets, *data* 16386 octets et *pile* 15870 octets. Est-ce que ce programme peut entrer dans l'espace disponible?

Il ne faut pas oublier que chaque région a ses propres pages. En effet, le texte vient d'un fichier exécutable et a une protection lecture/exécution, alors que la pile croit vers le bas et vient avec lecture et écriture, par exemple. Dans le cas qui nous intéresse, l'espace contient 64Kio/4Kio = 16 pages, 8 prises par le texte, 5 requises pour les données (16386 > 4 x 4096) et 4 requises pour la pile, ce qui est trop au total.

7.10 Une page peut-elle faire partie de l'espace de travail de deux processus en même temps?

Bien sûr, il est tout à fait possible pour deux processus d'utiliser le même code exécutable (programme ou librairie partagée), le même fichier attaché en mémoire ou la même zone de mémoire partagée.

7.11 Est-il possible qu'une page soit en lecture seulement pour un pocessus et en lecture et écriture pour un autre?

Oui, c'est tout à fait possible, l'information sur les accès permis est dans la table de pages de chaque processus. On peut imaginer un cas ou un processus maître s'occupe des mises à jour et des processus esclaves ne font que lire cette information.

7.12 Une instruction assembleur charge une valeur de 32 bits dans un registre sur une architecture semblable à celle du Intel Pentium. Combien de défauts de pages pourraient subvenir à cause de cette instruction?

Cette architecture n'ayant pas de contrainte d'alignement, il est possible que l'instruction tout comme la valeur de 32 bits soit à la frontière et s'étende sur deux pages. Ceci pourrait donc faire un total de 4 fautes de pages pour cette seule instruction.