TD 3

Gestion de la mémoire linéaire

semaine du 27/9/99

Objectifs : Etude de différents algorithmes de gestion d'une mémoire dans le cadre d'un système multiprogrammé.

Présentation

On se propose d'étudier plusieurs algorithmes de gestion d'une mémoire dans le cadre d'un système multiprogrammé où une tâche, pour s'exécuter, doit être présente en totalité et occupe un espace contigu en mémoire centrale (MC). La taille de la mémoire centrale est max. La partie résidente du système d'exploitation utilise la zone basse de la mémoire (commençant à l'adresse 0) ; la zone réservée aux processus utilisateurs commence donc à l'adresse ad_util.

Le système dispose d'une mémoire secondaire (MS). Pour effectuer des transferts entre MS et MC, on utilise la primitive transfert(sens,admc,adms,nb) où sens spécifie le sens du transfert (S: vers la MS; C: vers la MC), admc désigne l'adresse en MC du début du transfert, adms désigne l'adresse en MS du début du transfert et nb spécifie la longueur en mots du transfert.

Pour chaque tâche n, on utilise nbt(n): nombre de mots de la tâche; admc(n): adresse du premier mot de la tâche en MC; et adms(n): adresse du premier mot de la tâche en MS.

Note: On pourra utiliser des primitives de gestion de liste en décrivant simplement leur action mais sans les détailler.

Compactage

Pour protéger le système, on dispose d'un registre de base B et d'un registre de limite L. Les compilateurs génèrent des adresses à partir de 0. En mode utilisateur, un accès à l'adresse A ne sera autorisé que si $A \leq L$ et l'accès se fera à l'adresse B+A (où le registre B contient admc de la tâche en cours). En mode système, tous les accès sont autorisés et se font directement, les registres B et L ne peuvent être modifiés qu'en mode système.

On étudie une multiprogrammation sans va et vient, les processus qui ne sont pas chargés en MC ne sont pas prêts. Lorsqu'une tâche est soumise, on lui donne la première zone libre qui peut la contenir (first fit). S'il n'y a pas de zone libre assez grande pour la contenir, la tâche est mise en attente. On gère donc une liste des zones libres pointée par zones_libres triée par adresse croissante sur deb_zl (une adresse est de la forme : (deb_zl, fin_zl)) et une liste des tâches en attente pointée par attente.

1) Décrivez les actions à réaliser par le gestionnaire de mémoire lors d'une commutation.

2) On décide dans cette question d'avoir une allocation premier arrivé/premier servi pour les tâches soumises. Pour lutter contre la fragmentation externe, on décide de compacter la mémoire lorsque la première tâche en attente ne peut pas être servie mais qu'il y a assez de mémoire (fragmentée) pour la servir.

2.1) On suppose que la mémoire fait 512 Kmots dont les 128 premiers sont utilisés par la partie résidante du système. Dessiner dans le cas suivant l'état de la mémoire aux temps 8, 17, 19, 23.

Tâche	T1	T2	T3	T4	T5	T6	T7
Taille en Kmots	100	80	120	60	90	130	20
Instant de soumission	0	2	5	7	9	14	16
Instant de fin	12	24	18	20	22	27	25

2.2) Ecrire l'algorithme de compactage (algorithme naïf qui teste s'il y a assez de place mémoire globalement et commence le compactage jusqu'à dégager un espace mémoire suffisant pour la tâche). On utilisera la primitive move(dep,arr,nb) qui copie nb mots de l'adresse dep (départ) à l'adresse arr (arrivée).

2.3) Ecrire l'algorithme à mettre en œuvre lorsqu'une tâche n est soumise.

2.4) Ecrire l'algorithme à mettre en œuvre lorsqu'une tâche n se termine.

2.5) Expliquez pourquoi cette méthode convient pour une multiprogrammation en mode batch mais pas en temps partagé. Quelles modifications proposez-vous pour faire du temps partagé sans dégrader les performances ?

3) On décide maintenant de donner de la mémoire au premier processus en attente qui peut tenir dans la mémoire libre.

3.1) Dessiner la mémoire aux temps 17 et 19 en reprenant l'exemple du § 2.1.

3.2) Donnez les modifications à apporter aux algorithmes du § 2.

3.3) Montrez à l'aide d'un exemple qu'il peut y avoir famine.