UNIVERSITÉ BORDEAUX I — U.F.R. Mathématiques et Informatique
Session de Février 1994 — J.Bétréma

Maîtrise d'Informatique
Systèmes d'exploitation 1
(Durée: 2 heures)

Documents autorisés: notes de cours et de T.D., ouvrage de J.Beauquier et B.Bérard. Autres documents interdits.

Le sujet comporte un seul problème; les trois premières questions sont faciles, et vous rapporteront 8 points si elles sont traitées posément et clairement. Les questions 4 (4 points) et 5 (6 points) sont indépendantes l'une de l'autre. La question 6 est là pour garantir que personne ne s'ennuie.

On considère un algorithme d'allocation de blocs contigüs en mémoire, où la liste des tailles possibles est $t_1,\ t_2, \ldots, t_n$ (pour une mémoire de taille totale t_n), avec les t_i donnés par:

$$t_1 = 1,\ t_2 = 2,\ t_i = t_{i-1} + t_{i-2} \text{ pour } i > 2$$

soit $t_3 = 3,\ t_4 = 5, t_5 = 8, \ldots$.

On maintient une liste L_k de blocs libres, de taille t_k, pour chaque valeur de k; chaque bloc libre est repéré par son adresse de base. Au départ, toutes les listes sont vides, sauf L_n qui contient 0.

Algorithme d'allocation d'un bloc de taille t: soit k le plus petit entier tel que $t_k \geq t$, et soit p le plus petit entier supérieur ou égal à k tel que L_p soit non vide (s'il n'en existe pas, la requête est mise en attente) :

- Si $k = p$, on alloue le premier bloc de L_p (et on le retire de la liste); chaque liste libre est gérée par l'algorithme FIFO.
- Sinon on scinde ce bloc libre, de taille t_p, en deux blocs libres, de tailles t_{p-1} (pour la partie *gauche*) et t_{p-2} (pour la partie *droite*), et on ajuste les listes libres en conséquence. Les deux blocs créés sont dits *compagnons*, et on itère l'algorithme à partir du calcul de p.

Algorithme de libération: un bloc libéré est fusionné avec son *compagnon* si celui-ci est libre aussi, et ainsi de suite de façon récursive.

1. Etudier le partitionnement d'une mémoire de taille 34, obtenu suite aux requêtes d'allocation suivantes: $+4, +10, +2$; indiquer aussi sur votre copie quel est, après chaque requête, le contenu des listes libres non vides.

   Attention: appliquer l'algorithme rigoureusement, et expliquer les points délicats; des résultats approximatifs ne seront pas considérés comme corrects.

2. Quelle est dans cette situation (ie. après les requêtes de la question 1) la taille maximale d'une demande d'allocation acceptable par le système? Calculer le pourcentage de fragmentation interne. Commenter (brièvement!) ces résultats.

3. Ajouter les requêtes suivantes (le signe - indique une demande de libération) : $-4, +4, -2, -10, -4$, et les traiter comme en 1.

4. On suppose que les opérations sur les listes de blocs libres (insertion, suppression, scission d'un bloc en deux, fusion de deux compagnons) sont protégées par des sémaphores d'exclusion mutuelle, ce qui garantit la cohérence de ces listes. Reste-t-il des problèmes de synchronisation de processus, pour la gestion de la mémoire? Discuter sur des exemples précis.

5. On ajoute à chaque bloc B un compteur $g(B)$ géré comme suit:

   - $g(B) = 0$ pour le bloc B initial (de taille maximale t_n).
   - Lorsqu'un bloc libre B est scindé en un compagnon gauche B_g et un compagnon droit B_d, on pose:

   $$g(B_g) := g(B) + 1,\ g(B_d) := 0.$$

   (a) Enoncer la règle de calcul de g lorsqu'on fusionne deux compagnons (il y a une seule règle raisonnable : pourquoi?).

   (b) Donner les valeurs de $g(B)$ pour chacun des blocs qui interviennent dans les questions 1 et 3 (donner le détail des calculs, pour chaque requête).

   (c) Comment utiliser $g(B)$ pour calculer la base et la taille du compagnon d'un bloc B de base b et de taille t? En déduire l'algorithme exact de libération d'un bloc.

6. Ecrire en langage C les algorithmes d'allocation et de libération, en supposant disponibles les fonctions de gestion de listes et de sémaphores de votre choix.