Université Bordeaux I
Session de janvier 1999

U.F.R. Mathématiques et Informatique
Alain Griffault

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

Documents interdits

Avertissement

Les exercices sont indépendants
L'ensemble est long, la notation en tiendra compte. (Le barème est indicatif de l'importance respective des exercices).

1 Cours : 2 points

Question 1.1 *Donnez les définitions des fragmentations interne et externe. Pour chacune, donnez un moyen pour l'améliorer, et la raison pour laquelle votre solution n'est pas retenue.*

2 Interblocage : 4 points

Imaginez la politique suivante d'allocation de ressources. On autorise des requêtes et des libérations de ressources à tout moment. Si une requête ne peut pas être satisfaite parce que les ressources ne sont pas disponibles, nous examinons tous les processus bloqués en attente de ressources. S'ils possèdent les ressources désirées, on leur enlève donc ces ressources et on les donne au processus demandeur. On incrémente le vecteur des ressources pour lesquelles le processus bloqué attend afin d'inclure celles qu'on lui a enlevé.

Par exemple, considérez un système avec trois types de ressources et le vecteur `Disponible` initialisé à (4,2,2). Si le processus P_0 demande (2,2,1), il les obtient. Si P_1 demande (1,0,1), il les obtient. Ensuite, si P_0 demande (0,0,1), il est bloqué (ressource non disponible). Si maintenant P_2 demande (2,0,0), il obtient la ressource disponible (1,0,0) et celle allouée à P_0 (puisque P_0 est bloqué). Le vecteur `Tenu` de P_0 est décrémenté à (1,2,1) et son vecteur `Requête` incrémenté à (1,0,1).

Question 2.1 *Un interblocage peut-il se produire ? Si c'est le cas, donnez un exemple. Sinon, quelle condition nécessaire ne peut pas être satisfaite ?*

Question 2.2 *Le blocage indéfini (la famine) peut-il se produire ? Si c'est le cas, donnez un exemple. Sinon, expliquez brièvement pourquoi ?*

3 Algorithmes de remplacements de pages : 4 points

Question 3.1 *Construisez, pour une mémoire comportant 3 cadres initialement vides, des suites de références les plus simples (courtes) possibles pour lesquelles :*

1. *le nombre de défauts de pages est plus petit pour l'algorithme FIFO que pour LRU.*
2. *le nombre de défauts de pages est plus petit pour l'algorithme LRU que pour FIFO.*

Question 3.2 *Construisez une suite de références la plus simple possible qui met en évidence l'anomalie de Belady :* **pour l'algorithme FIFO, l'augmentation de la mémoire peut augmenter le nombre de défauts de pages.** *Vous pourrez chercher à continuer la suite* **0 1 2 3 0 1 4** *sur une mémoire constituée de 3 pages, puis de 4 pages.*

4 Synchronisation avec des sémaphores : 5 points

On dispose de processus `a, b, c` et `init` qui sont lancés au même instant. Le but est de contrôler l'ordonnancement des actions des processus `a, b` et `c`. Pour cela on dispose d'un langage de spécification comportant les opérations de concaténation séquentielle `'*'` et `'.'`, de choix exclusif `'+'` et de mise en parallèle `'||'`. Les processus `a, b` et `c` exécutent le code suivant :

```
Processus x() /* les  codes sont différents pour x = a, b ou c */
debut
   boucle { avantx ; actionsx ; apresx ; }
fin           /* actionx n'est pas forcément atomique */
```

Remarquons que si `avantx` et `apresx` sont vides pour un processus donné, celui-ci exécute indéfiniment `actionx`, et ce de façon séquentielle mais parallèlement aux autres processus.

Question 4.1 *Donnez, pour les spécifications suivantes (où x représente* `actionx` *) le code (qui est toujours une séquence de* `wait()` *et/ou de* `signal()`*) du processus* `init`*, et de* `avantx` *et* `apresx` *pour chaque processus* `a, b` *et* `c`.

1. `(a.b.c)* soit : a . b . c . a . b . c .` *etc*
2. `(a+b+c)* exemple : a . b . c . a . a . c . c . b` *etc*
3. `(a||b||c)* soit : a||b||c . a||b||c . a||b||c .` *etc*

Pour répondre, on utilisera le format suivant :

Langage de spécification : ...
Noms des sémaphores : ...
Code du processus init : ...

	a	b	c
avantx			
apresx			

Rappels :

```
type semaphore = (n : entier; l : liste de processus);
```

manipulée uniquement par les actions atomiques `wait` ($\equiv$ P) et `signal` ($\equiv$ V).

```
procedure wait(var S : semaphore)
debut
S.n := S.n - 1;
si (S.n < 0) alors debut
        ajouter(moi,S.l);
        block;
        fin
fin
```

```
procedure signal(var S : semaphore)
debut
S.n := S.n + 1;
si (S.n <= 0) alors debut
        enlever(P,S.l);
        wakeup(P);
        fin
fin
```

5 Exclusion mutuelle répartie : 5 points

Les horloges logiques définissent un temps $C(e)$ pour chaque événement e tel que :

1. Si a est avant b pour un processus X, alors $C(a) < C(b)$.
2. Si un processus X envoie un message M à un processus Y, alors $C(envoi(M)) < C(reception(M))$.

L'algorithme de Lamport permet l'obtention d'horloges logiques.

1. Pour chaque processus X incrémenter localement $C_X()$ pour chaque nouvel événement e.
2. Pour chaque processus X, pour chaque envoi de message M, ajouter l'estampille $C_X()$.
3. Lorsqu'un processus Y reçoit un message (M, E) ou M est un message et E une estampille, alors faire $C_Y() = max(C_Y(), E) + 1$

Rappel : Une relation R d'ordre strict (anti-réflexive, anti-symétrique et transitive) est :

totale ssi $\forall a, b\ : (a,b) \in R\ ou\ (b,a) \in R$.

partielle ssi $\exists a, b\ tel\ que\ : (a,b) \notin R\ et\ (b,a) \notin R$.

Soit $X = X_1, ..., X_N$ un ensemble de machine, soit $P = \{P_{i_j}\}$ un ensemble de processus (P_{i_j} s'exécute sur X_i) et soit $E = \{e^k_{i_j}\}$ un ensemble d'événements ($e^k_{i_j}$ est un événement du processus P_{i_j}).

Question 5.1 *Montrez que la relation R définie par : $((X_p, P_{p_j}, e^k_{p_j}), (X_q, P_{q_j}, e^k_{q_j})) \in R$ ssi $C_{P_{p_j}}(e^k_{p_j}) < C_{P_{q_j}}(e^k_{q_j})$ défini un ordre partiel sur $X \times P \times E$.*

Question 5.2 *Donnez une relation R' sur $X \times P \times E$ qui soit totale et qui respecte les horloges logiques.*

Soit l'algorithme réparti suivant pour le problème de l'exclusion mutuelle.

1. Si un processus P d'une machine X désire entrer en section critique, il envoie un message $< demande, C_P() >$ ($C_P()$ est l'horloge locale de P) à tous les processus de toutes les machines qui sont également concernés par cette section critique, puis il attend l'autorisation de chacun.

2. Lorsqu'un processus Q reçoit un message $< demande, E >$ de la part d'un processus P, il réagit ainsi :

   Non concerné il renvoie un message $< OK, C_Q() >$ à P.

   En section critique il attend sa sortie, puis envoie un message $< OK, C_Q() >$ à P.

   Concerné Si $(E, C_Q()) \in R$ alors il renvoie un message $< OK, C_Q() >$ à P. Sinon rien.

Question 5.3 *L'algorithme est-il correct si la relation R est celle définie par la relation d'ordre partielle sur $X \times P \times E$? Si oui, expliquez pourquoi, sinon donnez la propriété non satisfaite et un exemple de telle situation.*

Question 5.4 *L'algorithme est-il correct si la relation R est la votre qui définie une relation d'ordre totale sur $X \times P \times E$? Si oui, expliquez pourquoi, sinon donnez la propriété non satisfaite et un exemple de telle situation.*