Université Bordeaux I — U.F.R. Mathématiques et Informatique
Session de septembre 1999 — Alain Griffault

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

Documents interdits

Avertissement

Les exercices sont indépendants. Le barème est indicatif de l'importance respective des exercices.

1 Cours : 2 points

Dans le journal *Le Monde* du mercredi 24 mars 1999, on trouve la définition suivante :

Noyau : Contrairement à Windows ou à MacOS, qui sont, du point de vue de l'utilisateur, des logiciels monolithiques, Linux comporte un noyau qui rassemble l'ensemble des fonctions élémentaires de l'ordinateur, épaulé par de nombreux programmes complémentaires. On y accède en mode texte. Le noyau de Linux peut également recevoir une interface graphique pour améliorer l'ergonomie de l'ordinateur.

Question 1.1 *Relever les confusions dans cette définition, et donner votre définition du terme noyau.*

2 Le problème de la section critique : 5 points

Un chercheur [Hyman, Comm.ACM, vol 9, 1966] a proposé la solution suivante pour le problème de la section critique.

Deux processus P_0 et P_1 qui entrent en compétition pour l'accès en section critique partagent les variables suivantes :

```
/* initialisation */
int drapeau[2] = {0,0};
int tour = 0;
```

Le code du processus i (i=0 ou i=1).

```
Processus i :
debut
  drapeau[i] = 1;
  tant que (tour != i)
  debut
    tant que (drapeau[1-i] == 1) attendre;
    tour = i;
  fin;
  <Section Critique>
  drapeau[i] = 0;
fin
```

Question 2.1 *Cette solution est-elle correcte ? Justifier.*

Vous pourrez mettre des étiquettes de repère dans le texte du programme et construire l'automate des comportements pour les deux processus qui exécutent cet algorithme.

Vous pourrez arrêter la construction de cet automate dès que vous aurez mis en évidence l'existence d'un état "incorrect".

3 Synchronisation avec des sémaphores : 6 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           /* actionsx n'est pas forcément atomique */
```

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

Question 3.1 *Donnez, pour les spécifications suivantes (où x représente* `actionsx` *) 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)+(b.a))* exemple : a . b . a . b . a . b . b . a` *etc*
3. `((a||b).(b||a))* soit : a||b . b||a . a||b .` *etc*

Pour répondre, on utilisera le format suivant :

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

	a	b	c
avantx			
apresx			

Les processus `a, b` et `c` exécutent maintenant le code suivant :

```
Processus x() /* les  codes sont différents pour x = a, b ou c */
debut
   boucle { avant1x ; actionsx ; avant2x ; actionsx ; ... ; avantNx ; actionsx ;}
fin           /* actionsx n'est pas forcément atomique */
```

Question 3.2 *Donnez, pour la spécification suivante (où x représente* `actionsx` *) le code (qui est toujours une séquence de* `wait()` *et/ou de* `signal()`*) du processus* `init`*, et des* `avantIx` *pour chaque processus* `a, b` *et* `c`*.*

1. `((a||b).(b||c).(c||a))* soit : a||b . b||c . c||a . a||b .` *etc*

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
```

4 Ordonnancement de processus : 7 points

On considère un système de gestion de processus avec n niveaux de priorité, de 1 (le moins prioritaire) à n (priorité maximale). Chaque processus P est doté d'une priorité de base $P.b$ et d'une priorité courante $P.c$, égale à sa priorité de base lors de la création du processus. A chaque priorité i correspond une file d'attente f_i, gérée par l'algorithme FIFO, et qui contient les processus prêts tels que $P.c = i$. Le temps est divisé en quanta, et l'ordonnanceur respecte l'algorithme suivant :

1. lorsqu'un processus P est suspendu, il est placé dans une file d'attente :
   - des processus bloqués si la suspension est due à une entrée-sortie,
   - des processus prêts s'il a épuisé son quantum de temps ;
2. tout processus prêt P tel que $P.c = i < n$ a sa priorité courante incrémentée, et passe de la file f_i à la file f_{i+1}.
3. soit m le plus grand entier tel que f_m ne soit pas vide ; le processus élu est le premier de cette file ;

Question 4.1 *Quel est le rôle, à votre avis, de la seconde des règles de l'algorithme ?*

Question 4.2 *Lorsque le processus élu épuise son quantum de temps, dans quelle file d'attente, à votre avis, l'ordonnanceur doit-il le placer ?*

Question 4.3 *Lorsqu'un processus bloqué est réveillé par un événement extérieur, quel doit être, à votre avis, l'action de l'ordonnanceur ?*

On suppose que $n = 3$, et que quatre processus sont en compétition pour le processeur ; P_1 et P_2 ont pour priorité de base 3, P_3 a pour priorité de base 2, et P_4 a pour priorité de base 1 ; ces processus font uniquement des calculs, et épuisent donc à tour de rôle leur quantum de temps.

Question 4.4 *En utilisant votre réponse à la question 2, simulez l'action de l'ordonnanceur, jusqu'à détection d'un comportement périodique.*

Question 4.5 *Une fois le comportement du système devenu périodique, quelles sont les fréquences respectives d'élection des processus P_1 à P_4 ? Commentez le résultat.*

Question 4.6 *Proposez une modification simple de la règle d'incrémentation des priorités, pour augmenter l'avantage accordé aux processus les plus prioritaires.*

Remarques : Lorsque plusieurs réponses sont possibles, donnez celle qui vous semble la mieux adaptée, formulez-la de façon précise, et justifiez votre choix.