Université Bordeaux I
Session de septembre 1999

U.F.R. Mathématiques et Informatique
Alain Griffault

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

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.*

Réponse 1.1(2 points)

Le noyau est l'ensemble des procédures qui, en mode superviseur, traitent les déroutements ; il assure les fonctions de base : gestion des processus, de la mémoire, et des E/S de bas niveau. Les autres parties du système d'exploitation peuvent être réalisées sous formes de serveurs (exemples : NFS, X-Window). Unix (Linux aussi) a un gros noyau, on préfère aujourd'hui un micro-noyau qui gère aussi les messages et des serveurs (exemples : Mach, NT).

Il y a donc un certain nombres de confusions dans la définition donnée par le journal *Le Monde* :

noyau et systèmes d'exploitation : Windows NT et MacOS comportent comme Linux un noyau.

commandes utilisateur et instructions en mode superviseur Tous les systèmes offrent à l'utilisateur un ensemble de commandes et de programmes ; par contre aucun ne permet à l'utilisateur d'accéder directement à la gestion de la mémoire, à l'ordonnanceur,...

interface graphique et interpréteur de commandes l'interface utilisateur varie selon les systèmes. MacOS n'offre qu'une interface graphique. Windows NT offre toujours la possibilité (du fait de la compatibilité ascendante avec DOS) d'ouvrir une fenêtre DOS. Linux privilégie les entrées des commandes sous forme textuelle, mais il existe aujourd'hui des environnements pour Linux dans lesquels l'utilisateur peut "tout faire" à la souris.

logiciels monolithiques ? Les nouveaux systèmes d'exploitation basés sur un micro-noyau et sur l'envoi et la réception de messages sont plus facilement modulables et extensibles. Ils ne requièrent pas obligatoirement une recompilation du noyau à chaque ajout de services. Les versions récentes de Linux utilisent la notion de module pour un résultat similaire.

---

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.*

Réponse 2.1(5 points) La solution n'est pas correcte. Il est en effet possible que les deux processus soient simultanément en section critique comme le prouve la trace d'exécution suivante :

Processus 0 :	Processus 1 :
debut	debut
	drapeau[1] = 1;
	(tour != 1) est vrai
	(drapeau[0] == 1) est faux;
drapeau[0] = 1;	
(tour != 0) est faux	
<Section Critique>	tour = 1;
<Section Critique>	(tour != 1) est faux
<Section Critique>	<Section Critique>

Le graphe de la figure 2 qui représente tous les comportements possibles de cet algorithme montre également le problème. Les états cscs110 et cscs111 correspondent aux deux situations anormales. Ce graphe est obtenu en étiquetant le code des processus ainsi :

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

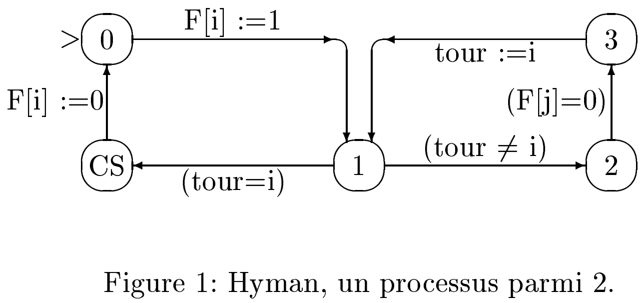

Figure 1: Hyman, un processus parmi 2.

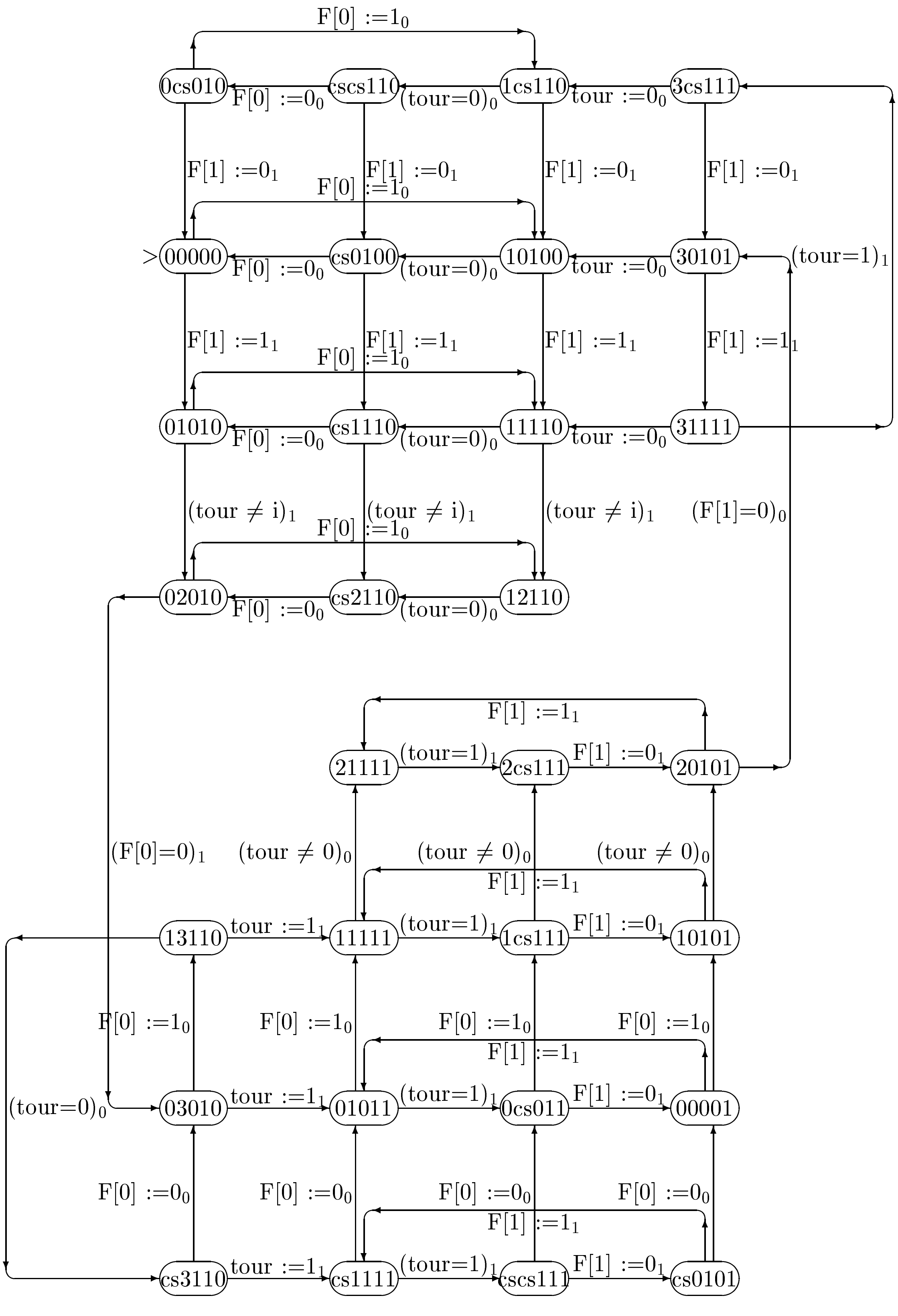

Figure 2: Hyman, 2 processus. (P_0,P_1,F[0],F[1],tour)

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*

Réponse 3.1(1,5+1,5+1 points)

Langage de spécification : (a.(b||c))*
Noms des sémaphores : A, B, C
Code du processus init : V(A)V(A)

	a	b	c
avantx	P(A)P(A)	P(B)	P(C)
apresx	V(B)V(C)	V(A)	V(A)

Langage de spécification : ((a.b)+(b.a))*
Noms des sémaphores : A, B, C, D
Code du processus init : V(A)V(B)V(D)

	a	b	c
avantx	P(A)P(D)	P(B)P(D)	P(C)
apresx	V(B)V(D)	V(A)V(D)	

Langage de spécification : ((a||b).(b||a))*
Noms des sémaphores : A, B, C
Code du processus init : V(A)V(B)

	a	b	c
avantx	P(A)	P(B)	P(C)
apresx	V(B)	V(A)	

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*

Réponse 3.2(2 points) Les codes diffèrent du fait de l'initialisation, mais il est facile de montrer plus de symétrie dans la solution.

```
Processus init()
debut
   V(A);V(B);V(B);
fin
```

```
Processus a()
debut
   boucle {
      P(A);
      actionsa ;
      V(B);V(C);P(A);P(A);
      actionsa ;
      V(B);
      }
fin
```

```
Processus b()
debut
   boucle {
      P(B);P(B);
      actionsb ;
      V(C);P(B);
      actionsa ;
      V(A);V(C);
      }
fin
```

```
Processus c()
debut
   boucle {
      P(C);P(C);
      actionsc ;
      V(A);P(C);
      actionsa ;
      V(A);V(B);
      }
fin
```

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 ?*

Réponse 4.1(1 points) Avec cette règle, l'algorithme est équitable ; c'est à dire qu'aucun processus n'attend indéfiniment pour être élu, et ce quelque soit les créations des processus au cours du temps.

La seconde règle évite donc la famine des processus de faible priorité.

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 ?*

Réponse 4.2(1 points) L'ordonnanceur place le processus P dans la file correspondant à sa priorité de base $P.b$.

L'autre choix possible est de le mettre dans la file correspondant à sa priorité courante $P.c$; mais avec ce choix, l'ordonnanceur va se comporter très rapidement comme une simple file FIFO.

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

Réponse 4.3(1 points) L'ordonnanceur calcule une priorité en fonction de l'événement extérieur. Cette priorité est généralement élevée soit pour permettre une interactivité forte (processus réveillé par entrée au clavier, à la souris, ...) ; soit pour améliorer le débit des entrées/sorties (processus réveillé par une interruption de fin de transfert).

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.*

Réponse 4.4(1 points) Nous supposons que les quatre processus sont créés simultanément, et que l'ordonnanceur débute par la troisième règle de son algorithme.

temps	f_1	f_2	f_3	élu
0	4	3	1,2	1
1a	4	3	2,1	
1b		4	2,1,3	
1c		4	2,1,3	2
2			1,3,2,4	1
3			3,2,4,1	3
4			2,4,1,3	2
5			4,1,3,2	4
6		4	1,3,2	1
7			3,2,1,4	3
8			2,1,4,3	2
9			1,4,3,2	1
10			4,3,2,1	4
11		4	3,2,1	3
12			2,1,4,3	2

La situation après le douzième cycle est identique à celle d'après le huitième.

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.*

Réponse 4.5(1 points) La période est de quatre, et au cours de cette période, chaque processus est élu une et une seule fois.

Sue cet exemple, les priorités ne sont d'aucune utilité.

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

Réponse 4.6(2 points) L'ordonnanceur n'effectue la seconde règle de l'algorithme que tous les k cycles. La période est certainement croissante avec k (non démontré)

Voici les évolutions de l'exemple précédent pour $k = 2$ et $k = 3$. L'astérix dans la première colonne indique les cycles au cours desquels l'ordonnanceur incrémente les priorités.

temps	f_1	f_2	f_3	élu
0	4	3	1,2	1
1a	4	3	2,1	
1c	4	3	2,1	2
2a	4	3	1,2	
2b*		4	1,2,3	
2c		4	1,2,3	1
3		4	2,3,1	2
4*			3,1,2,4	3
5		3	1,2,4	1
6*			2,4,1,3	2
7			4,1,3,2	4
8*		4	1,3,2	1
9		4	3,2,1	3
10*			2,1,4,3	2
11			1,4,3,2	1
12*			4,3,2,1	4
13	4		3,2,1	3
14*		4	2,1,3	2
15		4	1,3,2	1
16*			3,2,1,4	3
17		3	2,1,4	2
18*			1,4,2,3	1
19			4,2,3,1	4
20*		4	2,3,1	2
21		4	3,1,2	3
22*			1,2,4,3	1
23			2,4,3,1	2
24*			4,3,1,2	4
25	4		3,1,2	3
26*		4	1,2,3	1
27		4	2,3,1	2
28*			3,1,2,4	3

temps	f_1	f_2	f_3	élu
0	4	3	1,2	1
1a	4	3	2,1	
1c	4	3	2,1	2
2a	4	3	1,2	
2c	4	3	1,2	1
3a	4	3	2,1	
3b*		4	2,1,3	
3c		4	2,1,3	2
4		4	1,3,2	1
5		4	3,2,1	3
6*			2,1,4,3	2
7			1,4,3,2	1
8			4,3,2,1	4
9*		4	3,2,1	3
10		4,3	2,1,	2
11		4,3	1,2	1
12*			2,1,4,3	2

	période	fréquence P_1	fréquence P_2	fréquence P_3	fréquence P_4
$k = 1$	4	1/4	1/4	1/4	1/4
$k = 2$	24	7/24	7/24	6/24	4/24
$k = 3$	6	2/6	2/6	1/6	1/6

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