Maîtrise d'Informatique **17 Novembre 2000**

Systèmes d'exploitation **Devoir surveillé**

Exercice 1.

Trois approches du problème de la section critique existent :

1. les solutions purement logicielles (exemple : Peterson) ;
2. les solutions purement matérielles (exemple : fonction TestAndSet) ;
3. les solutions mixtes basées sur des appels systèmes implémentés grâce aux primitives matérielles (exemple : les sémaphores de Dijkstra)

Expliquer (en moins de 5 lignes) en quoi les deux premières solutions ne sont pas satisfaisantes pour un système d'exploitation.

Exercice 2.

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:

    drapeau[i]=1;
    tant que(tour != i)
       debut
          tant que(drapeau[1-i] == 1) attendre;
          tour = i;
       fin;
    <<section critique>>
    drapeau[i] = 0;
```

1. Décrire les différentes composantes des états de l'automate de comportements.
2. Placer les étiquettes de repère dans le texte du programme.
3. Construire l'automate des comportements. On pourra s'arrêter dès la détection d'un état «incorrect» vis à vis du problème de l'exclusion mutuelle (i.e. détection d'un état d'interblocage ou d'un état ne satisfaisant pas la propriété désirée).
4. Indiquer la plus simple suite d'actions de deux processus qui mène à cet état.

Exercice 3. Synchronisation de processus

On dispose de processus `A`, `B`, `C`, `D` et `init` qui sont lancer au même instant.

Les processus `A`, `B`, `C`, `D` exécutent le code suivant :

```
processus X(){
    while(1){
      avant_x;
```

```
      x;
      apres_x;
    }
}
```
où

- `X` est un de quatre processus `A`, `B`, `C`, `D`,
- `x` est la tâche exécutée par le processus `X` (la tâche dépend du processus : pour le processus `A` c'est la tâche `a`, pour `B` c'est la tâche `b` et ainsi de suite),
- `avant_x` et `apres_x` sont des opérations sur le sémaphores nécessaires à la synchronisations (et qui dépendent de processus).

L'ordre des tâches est spécifie à l'aide d'un langage de spécifications comportant des opération de concaténation séquentielle «·», de choix exclusif «+» et de mise en parallèle «‖» et d'itération séquentielle «*».

Le processus `init` servira à initialiser les sémaphores, il est composé d'une séquence de `wait` appropriée (on suppose que les sémaphores ont la valeur 0 à la création).

Donner pour la spécification

$$(((a+b)\|c)\cdot d)^*$$

le code de `init` et de `avant_x`, `apres_x` pour chaque processus `X`. Notez que le code de `init`, `avant_x` et `apres_x` est toujours une séquence de `wait` et `signal`.

Exercice 4. Donner les assignations produites par l'algorithme PCTER (plus court temps d'exécution restant) dans l'exemple suivant:

Tâche	Durée	Arrivée
A	5	0
B	2	2
C	7	3
D	2	6
E	3	12

Calculer le temps d'exécution moyen (rappel: le temps d'exécution d'une tâche est défini comme la différence entre la date de fin d'exécution et la date d'arrivée).
Le temps d'exécution moyen est-il inférieur ou supérieur si on applique l'algorithme du tourniquet?

Exercice 5. On considère un système de compagnons avec la taille des blocs 2^i octets, $i = 2,3,\ldots,12$. La taille minimale d'un bloc est 4 octets. La taille de la mémoire est de 4Ko.

Donner l'état de la mémoire après chacune des opérations suivante:

- allocation d'une zone A de 300 octets
- allocation d'une zone B de 100 octets
- allocation d'une zone C de 200 octets
- allocation d'une zone D de 200 octets
- allocation d'une zone E de 100 octets
- libération de la zone A
- libération de la zone E
- allocation d'une zone F de 600 octets
- libération de la zone C
- libération de la zone B

Exercice 6. (exercice complémentaire)

Un sémaphore binaire est un sémaphore dont le compteur a pour valeur soit 0 (pas de ressource disponible) soit 1 (une ressource disponible).

Ces sémaphores, dont les compteurs sont initialisés à 1 par la fonction `create_bin`, se manipulent via les fonctions `wait_bin` et `signal_bin`. On notera que `signal_bin(s)` n'a aucun effet lorsque le compteur de `s` vaut 1.

Compléter la procédure `wait` afin de simuler un sémaphore quelconque à l'aide 2 sémaphores binaires.

```
typedef struct sema
{
  semaphore_binaire a, mutex ;
  int cpt ;
} *semaphore ;

semaphore
create()
{
  semaphore s = malloc(sizeof(struct sema));
  s->a = create_bin();
  s->mutex = create_bin();
  s->cpt = 1; /* une seule ressource est disponible au départ */
}

void
signal(semaphore s)
{
  wait_bin(s->mutex);
  s->cpt++;
  signal_bin(s->a);
  signal_bin(s->mutex);
}

void
wait(semaphore s)
{
 ...
}
```

Corrigé de l'exercice 1.

Contrairement aux solutions basées sur des appels systèmes, les solutions purement matérielles ou purement logicielles génèrent de l'attente active.

Corrigé de l'exercice 2.

```
Processus i:

A:    drapeau[i]=1;
B:    tant que(tour != i)
       debut
C:          tant que(drapeau[1-i] == 1) attendre;
D:          tour = i;
       fin;
E:    <<section critique>>
F:    drapeau[i] = 0;
```

Les composantes de l'automate de comportement :
(ET_0, ET_1, `tour[0]`, `tour[1]`, `drapeau`), où ET_i l'étiquette (la position) du processus i.

état	état suivant		état symétrique
	processus 0	processus 1	
AA000	BA100	AB010	
BA100	EA100	BB110	
EA100	FA100	EB110	
FA100	AA000	FB110	
AB010	BB110	AC010	
BB110	EB110	BC110	
EB110	FB110	EC110	
FB110	AB010	FC110	
AC010	BC110	AD010	
BC110	EC110	BC110	
EC110	FC110	EC110	
FC110	AC010	FC110	
AD010	BD110	AE011	
BD110	ED110	BE111	
ED110	FD110	EE111	
AE011			EA100
BE111			EB110

Les deux processus peuvent se retrouver en même temps en section critique (état EE111) par la suite suivante d'actions : AA000 $\xrightarrow{1}$ AB010 $\xrightarrow{1}$ AC010 $\xrightarrow{1}$ AD010 $\xrightarrow{0}$ BD110 $\xrightarrow{0}$ ED110 $\xrightarrow{1}$ EE111.

Corrigé de l'exercice 3.

Trois sémaphores : `AB`, `C`, `D`.

Le processus `init` : `signal(AB), signal(C)`.

avant	tache	apres
`wait(AB)`	`a`	`signal(D)`
`wait(AB)`	`b`	`signal(D)`
`wait(C)`	`c`	`signal(D)`
`wait(D), wait(D)`	`d`	`signal(AB), signal(C)`

Corrigé de l'exercice 4.

Horloge	Tâche	Durée	Attente
2	A	3	
4	B	0	2
5	A	2	
8	D	0	2
11	A	0	11
15	E	0	3
22	C	0	19

Temps d'attente moyen: $37/5 = 7.2$. L'algorithme du Tourniquet nécessite un temps supérieur (ou égal) car le PCTER est optimal.

Corrigé de l'exercice 5.

1. requête $A(+300)$: $L_9 = (512)$,$L_{10} = (1024)$,$L_{11} = (2048)$, bloc alloué: 0, de taille 512.
2. requête $B(+200)$: $L_7 = (640)$,$L_8 = (768)$,$L_{10} = (1024)$,$L_{11} = (2048)$, bloc alloué: 512, de taille 128.
3. requête $C(+200)$: $L_7 = (640)$,$L_{10} = (1024)$,$L_{11} = (2048)$, bloc alloué: 768, de taille 256.
4. requête $D(+200)$: $L_7 = (640)$,$L_8 = (1280)$,$L_9 = (1536)$,$L_{11} = (2048)$, bloc alloué: 1024, de taille 256.
5. requête $E(+100)$: $L_8 = (1280)$,$L_9 = (1536)$,$L_{11} = (2048)$, bloc alloué: 640, de taille 128.
6. requête $A(-300)$: $L_8 = (1280)$,$L_9 = (0{,}1536)$,$L_{11} = (2048)$,
7. requête $E(-100)$: $L_7 = (640)$,$L_8 = (1280)$,$L_9 = (0{,}1536)$,$L_{11} = (2048)$,
8. requête $F(+600)$: $L_7 = (640)$,$L_8 = (1280)$,$L_9 = (0{,}1536)$,$L_{10} = (3076)$, bloc alloué: 2048, de taille 1024.
9. requête $C(-200)$: $L_7 = (640)$,$L_8 = (768{,}1280)$,$L_9 = (0{,}1536)$,$L_{10} = (3076)$, requête $B(-100)$: $L_7 = (512{,}640)$,$L_8 = (768{,}1280)$,$L_9 = (0{,}1536)$,$L_{10} = (3076)$,
10. fusion $(512{,}640) \Rightarrow L_8 = (512{,}768{,}1280)$,$L_9 = (0{,}1536)$,$L_{10} = (3076)$
11. fusion $(512{,}768) \Rightarrow L_8 = (1280)$,$L_9 = (0{,}512{,}1536)$,$L_{10} = (3076)$
12. fusion $(0{,}512) \Rightarrow L_8 = (1280)$,$L_9 = (1536)$,$L_{10} = (0{,}3076)$

Corrigé de l'exercice 6.

```
void
wait(semaphore s)
{
   wait_bin(s->a);
   wait_bin(s->mutex);
   s->cpt--;
   if (s->cpt > 0)
     signal (s->a);
   signal_bin(s->mutex);
}
```