UNIVERSITÉ DE ROUEN	ANNÉE 2004-2005
FACULTÉ DES SCIENCES	SYSTÈME D'EXPLOITATION
LICENCE INFORMATIQUE 3ème ANNÉE	FICHE DE TD N° 4

Segments de mémoire partagée

Exercice 1 Le but de cet exercice est l'étude de différentes solutions pour le problème d'accès à une section critique.

1) Quelles sont les conditions que doit vérifier une solution du problème de la section critique ?

Soient P1 et P2 deux processus s'exécutant en parallèle et accédant à une même section critique. Soit `libre` une variable partagée de type booléen initialisée à `vrai`. Considérons les codes des deux processus P1 et P2.

```
/* code de P1 */
while (1) {
<section non critique>
while (!libre)  ;
libre = 0;
<section critique>
libre = 1;
}
```

```
/* code de P2 */
while (1) {
<section non critique>
while (!libre)  ;
libre = 0;
<section critique>
libre = 1;
}
```

2) Cette solution réalise-t-elle l'exclusion mutuelle des deux processus ?

3) Existe-t-il une situation d'interblocage ?

On utilise maintenant une variable partagée `qui`, de type entier, initialisée par le numéro d'un processus (1 ou 2). Les codes de P1 et P2 sont :

```
/* code de P1 */
while (1) {
<section non critique>
while (qui != 1)  ;
qui = 1;
<section critique>
qui = 2;
}
```

```
/* code de P2 */
while (1) {
<section non critique>
while (qui != 2)  ;
qui = 2;
<section critique>
qui = 1;
}
```

4) Cette solution réalise-t elle l'exclusion mutuelle ?

5) Quels sont ses inconvénients ?

Dans cette troisième tentative on utilise un tableau `tbool` partagé de deux booléens initialisé à {0,0}. Les codes de P1 et P2 sont :

```
/* code de P1 */
while (1) {
<section non critique>
while (tbool[1])  ;
tbool[0] = 1;
<section critique>
tbool[0] = 0;
}
```

```
/* code de P2 */
while (1) {
<section non critique>
while (tbool[0])  ;
tbool[1] = 1;
<section critique>
tbool[1] = 0;
}
```

6) Montrer qu'il n'existe pas de situation de blocage.

7) Montrer que cette solution ne réalise pas l'exclusion mutuelle.

Changeons les codes de nos deux processus :

```
/* code de P1 */
while (1) {
<section non critique>
tbool[0] = 1;
while (tbool[1])  ;
<section critique>
tbool[0] = 0;
}
```

```
/* code de P2 */
while (1) {
<section non critique>
tbool[1] = 1;
while (tbool[0])  ;
<section critique>
tbool[1] = 0;
}
```

8) Montrer qu'il y a exclusion mutuelle.

9) Monter qu'il y a interblocage.

Cette dernière solution est une combinaison des solutions précédentes. Elle consiste à utilier un tableau de deux booléens `tbool[2] = {0, 0};` et un entier `qui`. Les codes de nos deux processus sont les suivants :

```
/* code de P1 */
while (1) {
<section non critique>
tbool[0] = 1;
qui = 0;
while (tbool[1] && qui == 0)  ;
<section critique>
tbool[0] = 0;
}
```

```
/* code de P2 */
while (1) {
<section non critique>
tbool[1] = 1;
qui = 1;
while (tbool[0] && qui == 1)  ;
<section critique>
tbool[1] = 0;
}
```

10) Montrer que cette solution est correcte. C'est-à-dire qu'elle réalise l'exclusion mutuelle et ne présente pas de cas d'interblocage.

Exercice 2 Le but de cet exercice est de simuler un calcul sur une machine parallèle disposant de n processeurs. Soit T un tableau de $n = 2^k$ entiers. On souhaite calculer la somme S donnée par la formule suivante :

$$S = \bigodot_{i=1}^{n} T(i) \tag{1}$$

où $\odot$ est une opération associative (par exemple : +, min, max, pgcd, $\times$, ...).

1) Proposer un programme utilisant n processus permettant de calculer S. Donner sa complexité en temps.

2) Peut-on réduire le nombre de processus tout en gardant la même complexité de calcul ? Si oui, expliquez comment.

Indication : Pour la synchronisation des processus on utilisera soit l'attente actice ou les appels systèmes bloquants tels que wait, waitpid, ...

Exercice 3 Un tableau `T` de $2N$ éléments est partagé en deux moitiés : on souhaite regrouper dans la première moitié les N plus grands éléments et dans la seconde les N plus petits. On se propose de donner une solution au problème en utilisant deux processus partageant le tableau en mémoire : l'un accède à la première moitié et recherche le plus petit élément, et l'autre recherche dans la seconde moitié le plus grand élément. Lorsque les deux processus ont terminé leur recherche, il y a éventuellement échange des deux éléments et itération du mécanisme.

Création d'un segment

```
int ids; /* identificateur du segment */
ids = shmget(IPC_PRIVATE, N*sizeof(int), IPC_CREAT|0666);
```

Attachement du segment

```
int *T;   /* adresse d'attachement */
T = shmat(ids, NULL, 0);
```

Détachement du segment

```
shmdt(T);
```

Suppression du segment

```
shmctl(ids, IPC_RMID, NULL);
```