Faculté des Sciences & Techniques — Université de Limoges

Licence 3ème année

Programmation Concurrente

Examen – mars 2023

Durée : 1h30 — Documents autorisés

Threads & Sémaphores — 6 points

1 – 6pts Le responsable d'un parking disposant de plusieurs entrées équipées de **portail d'accès informatisé** et géré par un même **système central** vous contacte.

Il voudrait disposer d'une simulation logicielle pour étudier **indépendemment chacun** des fonctionnements suivants :

a. Avec quel(s) **mécanismes de programmation concurrente** allez vous autoriser l'accès ou non au parking à un véhicule ? *(1pt)*
Chaque véhicule sera considéré comme une thread dans le programme .

b. La solution doit garantir qu'il **ne laisse pas entrer** dans son parking plus de n voitures pour les n places disponibles. *(1pt)*

c. Chaque voiture dispose d'un **emplacement de parking réservé**. *(1pt)*
La solution doit garantir qu'il n'y ait pas plus d'une voiture sur l'emplacement qui lui est réservé.

d. Une **offre famille** a été définie : un emplacement peut être associé à deux voitures de la même famille et seule une de ces deux voitures peut accèder à la place de parking. *(1pt)*

e. Une **offre moto** a été définie : un emplacement peut être utilisé par deux motos simultanément ou par une seule voiture. *(1pt)*

f. Une **offre « famille & moto »** : un emplacement est associé à une seule famille qui peut y ranger soit deux motos soit une voiture. *(1pt)*

Vous respecterez les consignes suivantes :

- ⋆ Chaque solution aux questions a) à e) est indépendante l'une de l'autre.
- ⋆ Vous donnerez le code correspondant :
  - ◇ à l'occupation de la place
  - ◇ à sa libération.
- ⋆ Si un véhicule, moto ou voiture, n'a pu obtenir une place, il est « mis en attente » pour être prioritaire lorsque la place se libérera.

Problématiques de la programmation concurrente et asynchrone — 6 points

2 – 6pts

a. Comment échanger le **plus rapidement** possible des données entre deux processus ? *(2pts)*
Comment tenir compte de la **synchronisation** ?
Vous justifierez votre réponse.

b. Quelles sont les **similarités** et les **différences** entre « *signaux* » et « *sémaphores* » ? *(2pts)*
Vous détaillerez votre réponse

c. Est-ce que le **problème d'équité** est plus simple à résoudre en programmation asynchrone ou en programmation à base de threads Posix ? *(2pts)*
Vous justifierez votre réponse.

Création de processus avec fork — 3 points

3 – Soit le programme C suivant :

3pts

```
 1 #include <stdio.h>
 2
 3 void traitement(int a, int b)
 4 {
 5    printf("-->%d\n", a*b);
 6    if (a)
 7    {
 8       if (fork())
 9       {
10          traitement(a-1, 2);
11       }
12       else
13       {
14          traitement(a-1, 3);
15       }
16    }
17 }
18
19 int main()
20 {
21    traitement(3,1);
22 }
```

Que va-t-il afficher lors de son **exécution** ?

Modélisation — 5 points

4 – Soient les 4 threads suivantes, s'exécutant chacune **une seule fois** :

5pts

```
thread_A {
 sem_wait(s1);
 x = x * y;
 sem_post(s3);
}
```

```
thread_B {
 sem_wait(s1);
 y = x + y;
 sem_post(s2);
}
```

```
thread_C {
 sem_wait(s2);
 y = y + 5;
 sem_post(s3);
}
```

```
thread_D {
 sem_wait(s2);
 x = x * 2;
 sem_post(s2);
}
```

Elles utilisent pour leur synchronisation trois sémaphores `s1`, `s2` et `s3`.

a. Quelles sont les **différentes valeurs possibles** des variables `x` et `y` à la fin de l'exécution complète des quatres *threads* avec les valeurs d'initialisation suivantes : *(3pts)*

Sémaphore	s1	s2	s3
valeur initiale	1	0	0

variable	x	y
valeur initiale	0	1

b. Qu'est-ce que cela change avec les valeurs d'initialisation suivantes : *(2pts)*

Sémaphore	s1	s2	s3
valeur initiale	2	1	1

variable	x	y
valeur initiale	0	1