Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Licence 3ème année

Parallélisme I

Corrections – juin 2014

Durée : 2h — Documents autorisés

Threads & Sémaphores – (12 points)

1 – 10pts Une société de production de série télévisée vous contacte afin de lui fournir un système réaliste pour la gestion d'une prison de haute sécurité futuriste qu'elle pourra ensuite utiliser dans l'écriture de ses scénarios, et même peut-être l'utiliser comme sujet principal pour une série.

Le scénario de base pour la prison est le suivant :

- chaque prisonnier est muni d'un bracelet qu'il ne peut enlever et qui va embarquer un ordinateur exécutant le programme que vous allez concevoir ;
- il existe deux classes de prisonniers A et B qui ont des autorisations d'accès différentes aux différentes parties de la prison :
  - la classe A de 5 prisonniers très astucieux et capables de s'évader qui doivent être gérés avec des restrictions ;
  - la classe B de 30 prisonniers moins astucieux ;
- la prison possède :
  - 15 cellules : 5 individuelles et 10 pouvant recevoir 3 prisonniers ;
  - un court de tennis pour un ou deux joueurs (quand un joueur est seul il s'entraîne contre un mur de rebond) ;
  - un réfectoire de 35 places ;
  - un lieu extérieur de promenade limité à 5 prisonniers au plus ;

Dans le fonctionnement normal de la prison, chaque prisonnier se voit attribuer une cellule unique identifiée par son numéro de 1 à 5 pour les individuelles et de 6 à 15 pour celles de 3. À la fin de la journée les prisonniers doivent aller obligatoirement dans leur cellule et ne pas se trouver dans les parties communes.

Chaque cellule n'est accessible qu'au prisonnier dont le bracelet l'autorise.

Questions :

a. *(2pts)* pour le premier scenario, les producteurs envisagent de montrer le fonctionnement de la prison dans son mode normal :
   - chaque prisonnier de classe A accède uniquement à sa cellule individuelle ;
   - les prisonniers de classe B accède par groupe de 3 à la même cellule qu'ils partagent ;

   Décrivez votre solution en termes de Sémaphores pour gérer l'accès des prisonniers de classe A et B à leur cellules respectives.
   Tout d'abord, on va considérer que le système «bracelet/porte» *permet de bloquer l'ouverture de la cellule pour un prisonnier qui n'a pas le droit d'y accéder. Ensuite, il faut décider du mécanisme utilisé pour ouvrir/fermer une cellule :*
   - *pour ouvrir on va considérer que le logiciel de la porte* ***prend un jeton*** *d'une sémaphore gérée par le bracelet ;*
   - *pour fermer la porte, la porte* ***libère le jeton*** *de la sémaphore gérée par le bracelet.*

   Pour répondre à la question, il faut une sémaphore associée à chaque cellule :
   - *une sémaphore pour chaque cellule des prisonniers de classe A : S_1 à S_5 ;*
   - *une sémaphore pour chaque cellule des prisonniers de classe B : S_6 à S_{15} ;*

b. *(2pts)* Pour un autre scenario, les producteurs veulent montrer le fonctionnement du court de tennis :
   - deux prisonniers de classe B peuvent avoir accès ensemble au court ;
   - un seul prisonnier de classe A peut accèder au court de tennis ;

   Décrivez votre solution en termes de Sémaphores pour gérer l'accès des prisonniers de classe A et B au court de tennis. *Pour gérer le fonctionnement du court de tennis, on va utiliser deux sémaphores gérées par la porte d'accès au court de tennis :*
   - *une sémaphore S_{tennis_A} : pour permettre l'accès à un prisonnier de classe A ;*

◇ *une sémaphore S_{tennis_B} : pour permettre l'accès à un prisonnier de classe B ;*

◇ *une semaphore Mutex ;*

Lors de l'inialisation, $S_{tennis_A} \longleftarrow 1$, Mutex $\longleftarrow 1$ et $S_{tennis_B} \longleftarrow 2$.

L'entrée d'un prisonnier se fait de la manière suivante :

◇ *pour un prisonnier de classe A :*

```
1 P(STennisA)
2 Jouer au tennis seul
3 V(STennisA)
```

pour un prisonnier de classe B :

```
1  P(STennisB)
2  P(Mutex)
3  NBJoueurB ++
4  Si (NBJoueur == 1)
5  { P(STennisA) }
6  V(Mutex)
7
8  Jouer au tennis tout seul ou à
9  deux
10
11 P(Mutex)
12 NBJoueurB --
13 Si (NBJoueur == 0)
14 { V(STennisA) }
15 V(Mutex)
16 V(STennisB)
```

*La solution est dérivée du problème des «*lecteurs/rédacteur*» en limitant le nombre de «*lecteur*» à deux.*

c. Le scénario suivant porte sur le lieu de promenade dont l'accès est réglementé de la façon suivante : *(2pts)*

◇ seul un seul prisonnier de classe A peut s'y trouver simultanément avec d'autres prisonniers de classe B ;

◇ les prisonniers de classe B peuvent y avoir accès comme ils veulent à condition qu'il n'y ait pas plus de 5 prisonniers ;

Décrivez votre solution en termes de Sémaphores pour gérer l'accès des prisonniers de classe A et B à la promenade.

Il faut :

◇ *une sémaphore pour le lieu de promenade $S_{promenade} \longleftarrow 5$;*

◇ *une sémaphore pour n'avoir qu'un prisonnier de classe A $S_{promA} \longleftarrow 1$;*

Ce qui donne :

pour un prisonnier de classe A :

```
1 P(SpromA)
2 P(Spromenade)
3 Se promener
4 V(Spromenade)
5 V(SpromA)
```

pour un prisonnier de classe B :

```
1 P(Spromenade)
2 Se promener
3 V(Spromenade)
```

Est-ce que votre solution est «équitable» ?

Oui : grâce à la gestion des sémaphores, chaque prisonnier de classe B qui ne peut obtenir ce qu'il demande va s'endormir et sera réveiller lorsqu'il y aura une place pour lui.

Non : un prisonnier de classe A doit d'abord être seul de sa classe, puis essaye d'obtenir une place. Un autre prisonnier de classe A devra attendre pour obtenir S_{promA}, mais ne sera pas prioritaire par rapport à l'accès au lieu de promenade par rapport aux prisonniers de classe B.

d. Le fonctionnement du réfectoire est le suivant : tous les prisonniers peuvent y avoir accès librement. Décrivez votre solution en terme de Sémaphores pour gérer l'accès des prisonniers de classe A et B au réfectoire. *(2pts)*

Il suffit d'une seule sémaphore $S_{réfectoire} \longleftarrow 35$ et chaque prisonnier se comporte de la manière suivante :

```
1 P(Sréfectoire)
2 Manger
3 V(Sréfectoire)
```

e. Pour un nouveau scenario, une restriction a été mise en place suite à une tentative d'évasion :

- le réfectoire n'est accessible simultanément qu'aux prisonniers de la même classe : un prisonnier d'une classe peut entrer dans le réfectoire uniquement si le réfectoire est vide ou occupé par des prisonniers de même classe.

(2pts)

Décrivez votre solution en termes de Sémaphores pour gérer l'accès des prisonniers de classe A et B à la promenade.

Il faut une sémaphore pour les prisonniers de classe A, $S_{réfectoire_A} \longleftarrow 0$ et une sémaphore pour les prisonniers de classe B, $S_{réfectoire_B} \longleftarrow 0$. Il faut également une sémaphore Mutex.

pour un prisonnier de classe A :

```
P(Mutex)
Si ((NBPrisonnierA == 0)
   ET (NBPrisonnierB == 0)
{
   NBPrisonnierA ++
   Pour i allant de 1 à 5
      { V(RéfectoireA) }
}
Sinon
{
   NBPrisonnierA ++
}
V(Mutex)
P(SRefectoireA)
Manger
V(SRefectoireA)
P(Mutex)
NBPrisonnierA --
Si (NBPrisonnierA == 0)
{
   Pour i allant de 1 à 5
      { P(RéfectoireA) }
}
Si (NBPrisonnierB != 0)
{
   Pour i allant de 1 à 30
      { V(RéfectoireB) }
}
V(Mutex)
```

pour un prisonnier de classe B :

```
P(Mutex)
Si ((NBPrisonnierA == 0)
   ET (NBPrisonnierB == 0)
{
   NBPrisonnierB ++
   Pour i allant de 1 à 30
      { V(RéfectoireB) }
}
Sinon
{
   NBPrisonnierB ++
}
V(Mutex)
P(SRefectoireB)
Manger
V(SRefectoireB)
P(Mutex)
NBPrisonnierB --
Si (NBPrisonnierB == 0)
{
   Pour i allant de 1 à 30
      { P(RéfectoireB) }
}
Si (NBPrisonnierA != 0)
{
   Pour i allant de 1 à 5
      { V(RéfectoireA) }
}
V(Mutex)
```

Est-ce que votre solution est «équitable» ?

Au tout début, lorsque la salle est vide, n'importe quelle classe de prisonniers est capable de réserver la salle : le premier prisonnier libère les accès aux autres prisonniers de sa classe.

*Si des prisonniers de la même classe se succèdent sans interruption, le **dernier** reprend les autorisations (reprend tous les jetons de la sémaphore) des autres prisonniers de la sa classe.*

À ce moment deux possibilités :

- *il n'y a pas de prisonniers de l'autre classe qui a essayé d'aller dans le réfectoire auquel cas on se ramène au début ;*
- *il y a des prisonniers de l'autre classe qui attendent l'accès et il libère des jetons pour cette autre classe.*

A priori, s'il y a succession de prisonniers de classe A puis de classe B, il y a alternance obligatoire, puisque le dernier d'une classe supprime les autorisations de sa classe et donne celles de l'autre classe. On peut penser avoir une solution équitable.

2 – Questions sur l'utilisation des threads :

2pts a. Pourquoi y-a-t-il des **risques de corruption** de données ? *(0,5pt)*

Parce que deux threads peuvent accèder à une même variable pour la modifier simultanément : le résultat sera imprévisible

b. Comment est partagée la «**pile ou stack**» entre les threads ? *(0,5pt)*

La pile initiale du processus sans thread est répartie entre les différentes threads : chaque thread possède sa propre pile.

Questions sur l'utilisation des sémaphores :

d. Comment sont réglés les problèmes de **famine** et d'**équité** ? *(1pt)*

Le problème de la famine est réglé par l'utilisation d'une file d'attente : toute demande de Sémaphore est mémorisée et sera traitée.

Le problème de l'équité est résolu par la file d'attente : chaque demande est traitée dans l'ordre d'arrivée.

Signaux & tubes & fork – (8 points)

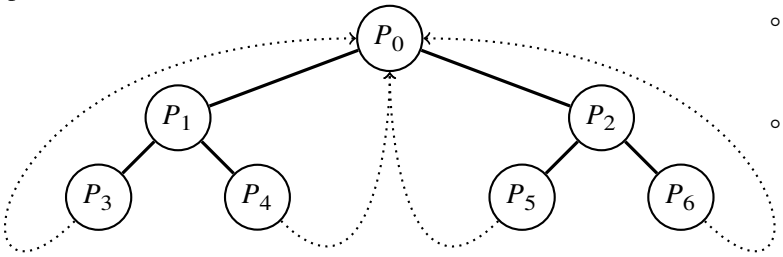
3 – 2pts Soit le programme C suivant :

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

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

```
-->3
-->4
-->3
-->2
-->3
-->0
-->3
-->3
-->3
```

4 – 6pts On veut mettre en place une structure **aborescente** de travail avec des tubes permettant de communiquer entre processus d'un **niveau** de l'arbre vers le suivant :



- le trait en gras indique un «*tube*» de communication ;
- le trait pointillé indique l'envoi d'un «signal».

1. un processus P_0 crée 2 fils $\{P_1, P_2\}$;
2. il crée un tube avec ses fils P_1 et P_2 permettant de communiquer du père vers le fils ;
3. le fils P_1 va :
   - ◇ créer 2 fils $\{P_3, P_4\}$;
   - ◇ établir un tube permettant de communiquer avec chacun de ses fils (de P_1 vers P_3 et de P_1 vers P_4) ;
   - ◇ P_3 et P_4 vont, après la mise en place du tube avec leur parent, envoyer un **signal à P_0** ;
4. le fils P_2 va :
   - ◇ créer 2 fils $\{P_5, P_6\}$;
   - ◇ établir un tube permettant de communiquer avec chacun de ses fils (de P_2 vers P_5 et de P_2 vers P_6) ;

- ◇ P_5 et P_6 vont, après la mise en place du tube avec leur parent, envoyer un **signal à P_0** ;

5. lorsque le père reçoit les 4 signaux (un de chacun de ses «*petits fils*»), il exécutera la fonction `void travail_arbre()` qui sera fournie dans une bibliothèque externe à votre programme.

Questions :

a. Pourquoi est-il nécessaire d'envoyer un **signal** depuis les «*petits fils*» vers le parent avant d'exécuter la fonction `travail_arbre` ? *(1pt)*
Pour synchroniser le travail du parent avec la création effective des petits enfants.

b. Comment les processus enfants de P_1 et P_2 vont pouvoir connaître le *pid* de P_0 ? *(1pt)*
Grâce à l'instruction `getppid()` ou en mémorisant le pid du parent dans une variable partagée entre les différents enfants.

c. Est-il possible d'avoir un *code* similaire pour le travail de P_1 et P_2 ? *(1pt)*
oui.

d. Écrire le programme `C` réalisant **uniquement** la création des 7 processus P_i de manière à permettre le travail demandé : *(3pts)*
- ◇ vous n'avez pas à créer de tubes, ni à envoyer de signal ;
- ◇ vous devez gérer les Pids (mémoriser, partager) pour rendre l'intégration du code des signaux et tubes possible.

```
1  #include <stdio.h>
2
3  int main()
4  {
5          int pid_p0 = getpid();
6
7          if (fork() == 0)
8          { /* P1 */
9                  if (fork() == 0)
10                 { /* P3 */
11                         ...
12                 }
13                 if (fork() == 0)
14                 { /* P4 */
15                         ...
16                 }
17         }
18         if (fork() == 0)
19         { /* P2 */
20                 if (fork() == 0)
21                 { /* P5 */
22                         ...
23                 }
24                 if (fork() == 0)
25                 { /* P6 */
26                         ...
27                 }
28         }
29
30 }
```