Université Bordeaux I
U.F.R. Mathématiques et Informatique

IUP MIAGe 3^{eme} année
Année 1997-1998

Vendredi 16 janvier, 14 heures - 16 heures
Tous documents autorisés.

Programmation Parallèle
Alain Griffault

Nom : Prénom :

Exercice 1 (7 points) : Ordonnancement de tâches avec des sémaphores

Un sémaphore S, à attente active, est un entier muni de deux primitives $P(S)$ et $V(S)$ qui permettent de le manipuler de façon atomique, P permet d'acquérir une ressource protégée par un sémaphore (lorsque l'entier est > 0) tout en décrémentant de 1 la valeur du sémaphore, et V permet de la libérer (en augmentant de 1 la valeur du sémaphore). Plus formellement :

P(S)	V(S)
tant que $S \leq 0$ faire rien $S := S - 1$	$S := S + 1$

Les sémaphores peuvent être utilisés pour garantir des ordonnancements entre tâches. Une première solution consiste à utiliser un sémaphore (initialisé à 0) par arête dans le graphe d'ordonnancement des tâches.

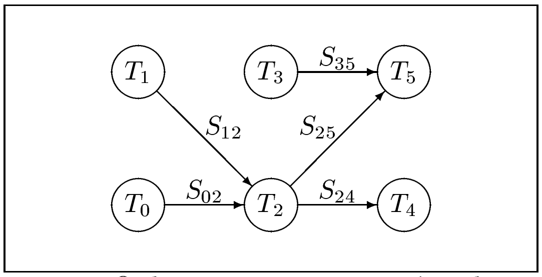

Figure 1: Ordonnancement avec 5 sémaphores.

Avec ces sémaphores, l'ordonnancement est obtenu, si chaque tâche exécute les codes suivants.

T_0	T_1	T_2	T_3	T_4	T_5
code V(S_{02})	code V(S_{12})	P(S_{02}) P(S_{12}) code V(S_{24}) V(S_{25})	code V(S_{35})	P(S_{24}) code	P(S_{25}) P(S_{35}) code

Question 1 *Expliquer en quelques lignes le fonctionnement de la tâche T_2.*

Une deuxième solution consiste à utiliser un sémaphore pour chaque tâche dépendante dans le graphe.

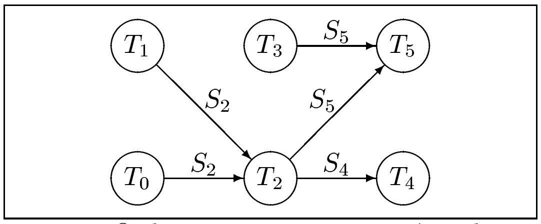

Figure 2: Ordonnancement avec 3 sémaphores.

Avec ces sémaphores, l'ordonnancement est obtenu, si chaque tâche exécute les codes suivants.

T_0	T_1	T_2	T_3	T_4	T_5
code V(S_2)	code V(S_2)	P(S_2) code V(S_4) V(S_5)	code V(S_5)	P(S_4) code	P(S_5) code

La deuxième solution diminue le nombre de sémaphores nécessaires, et diminue la nombre d'actions P et V nécessaires. En contre partie, les initialisations des valeurs des sémaphores ne sont pas toujours 0.

Question 2 *Donner les valeurs initiales des trois sémaphores, afin de garantir l'ordonnancement des tâches.*

Sémaphore	S_2	S_4	S_5
Valeur initiale			

Un projet complexe est constitué de 4 tâches élémentaires qui se répètent indéfiniment. Les contraintes d'antériorités sont données par le graphe infini suivant :

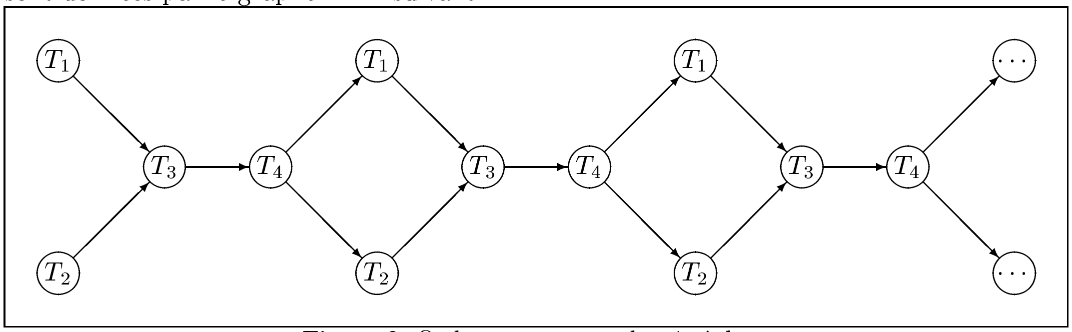

Figure 3: Ordonnancement des 4 tâches.

Question 3 *En utilisant l'une ou l'autre des deux méthodes précédentes, placez aux bons endroits dans chaque tâche, les primitives P et V nécessaires.*

T_1	T_2	T_3	T_4

begin
 	
while true {
 	
code
 	
}
 	
end
	begin
	
	while true {
	
	code
	
	}
	
	end
		begin
		
		while true {
		
		code
		
		}
		
		end
			begin
			
			while true {
			
			code
			
			}
			
			end

Question 4 *Pour chacun des sémaphores utilisés, donner les valeurs initiales.*

Sémaphore									
Valeur initiale									

Exercice 2 (7 points) : Le problème de l'exclusion mutuelle

Ce problème est relativement simple. Il s'agit de garantir à un processus qu'il sera le seul à un instant donné, à exécuter une portion critique de son code. En d'autres termes, plusieurs processus exécutent le programme suivant en parallèle :

```
Algorithme d'un processus
begin
while true do
  begin
  < section non critique >
  (* essayer d'entrer en section critique *)
  < section critique >
  (* sortir de la section critique *)
  end
end
```

En 1966, H. Hyman a publié dans le cas de deux processus une solution qui utilise les trois variables globales `flag[0] : booleen, flag[1] : booleen, token : 0..1`, initialisées respectivement à `false, false, 0`.

```
Algorithme du processus I
begin
while true do
  begin
  < section non critique >
  flag[I] := true;
  while token <> I do
    begin
    while flag[1-I] = true do
      begin
      end
    token := I
    end
  < section critique >
  flag[I] := false;
  end
end
```

Question 5 *Montrez, en donnant une exécution (à chaque instant, un seul des processus effectue une action) possible de ce programme, que cette solution est fausse du fait qu'il est possible que les deux processus soient simultanément en section critique.*

Instant	Processus 0	Processus 1	flag[0]	flag[1]	token
0 (initial)			false	false	0
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					

Exercice 3 (7 points) : Les philosophes ...

Voici des sources java pour le problème des philosophes mangeurs de spaghettis.

```
public class Fourchettes {
  private boolean[] forks; // false = non tenue

  Fourchettes(int size) {
    forks = new boolean[size] ;
    for (int i=0; i<size; i++) { forks[i] = false ;}
  }

  public void prendre(String who, int nb) {
    while (forks[nb] == true) {};
    forks[nb] = true ;
    System.out.println(who + " tient la fourchette " + nb);
  }

  public void poser(String who, int nb) {
    forks[nb] = false ;
    System.out.println(who + " ne tient plus la fourchette " + nb);
  }
}

class Philosophe extends Thread {
  private int threadId, nb;
  private static Fourchettes fourchettes;

  Philosophe(int i, int size, Fourchettes four) {
     setName(``Phil_''+i+'' ``);
     threadId = i;
     nb = size;
     fourchettes = four;
  }

  public void run() {
     while (true) {
       System.out.println(getName() + " pense.");
       fourchettes.prendre(getName(), threadId);
       fourchettes.prendre(getName(), (threadId +1) % nb);
       System.out.println(getName() + " mange.");
       fourchettes.poser(getName(), threadId);
       fourchettes.poser(getName(), (threadId +1) % nb);
    }
  }
}

class Spaghetti extends Thread {
  public static Fourchettes fourchettes;
  public static Philosophe[] philosophes;

  public static void main(String []args) {
    int size = Integer.parseInt(args[0]);
    fourchettes = new Fourchettes(size);
    philosophes = new Philosophe[size];
    for (int i=0; i<size; i++)
       philosophes[i] = new Philosophe(i,size,fourchettes).start();
  }
}
```

Question 6 *Expliquez brièvement les problèmes potentiels du programme précédent.*

Question 7 *Que pensez-vous de la solution suivante qui modifie la classe Fourchettes.*

```
public class Fourchettes {
  private boolean[] forks; // false = non tenue

  Fourchettes(int size) {
    forks = new boolean[size] ;
    for (int i=0; i<size; i++) { forks[i] = false ;}
  }

  synchronized public void prendre(String who, int nb) {
    while (forks[nb] == true) {
       try {wait();}
       catch (InterruptedException e) {}
    }
    forks[nb] = true ;
    System.out.println(who + " tient la fourchette " + nb);
  }

  synchronized public void poser(String who, int nb) {
    forks[nb] = false ;
    System.out.println(who + " ne tient plus la fourchette " + nb);
    notify();
  }
}
```

Question 8 *Proposez votre solution en décorant le source initial.*

```
public class Fourchettes {
  private boolean[] forks; // false = non tenue

  Fourchettes(int size) {
    forks = new boolean[size] ;
    for (int i=0; i<size; i++) { forks[i] = false ;}
  }

public void prendre(String who, int nb) {
    while (forks[nb] == true) {

    };
    forks[nb] = true ;
    System.out.println(who + " tient la fourchette " + nb);
  }

  public void poser(String who, int nb) {
    forks[nb] = false ;
    System.out.println(who + " ne tient plus la fourchette " + nb);
  }
}

class Philosophe extends Thread {
  private int threadId, nb;
  private static Fourchettes fourchettes;

  Philosophe(int i, int size, Fourchettes four) {
     setName(‘‘Phil_i’’+i+’’ ‘‘);
     threadId = i;
     nb = size;
     fourchettes = four;
  }

  public void run() {
     while (true) {
       System.out.println(getName() + " pense.");
       fourchettes.prendre(getName(), threadId);
       fourchettes.prendre(getName(), (threadId +1) % nb);
       System.out.println(getName() + " mange.");
       fourchettes.poser(getName(), threadId);
       fourchettes.poser(getName(), (threadId +1) % nb);
    }
  }
}

class Spaghetti extends Thread {
  public static Fourchettes fourchettes;
  public static Philosophe[] philosophes;

  public static void main(String []args) {
    int size = Integer.parseInt(args[0]);
    fourchettes = new Fourchettes(size);
    philosophes = new Philosophe[size];
    for (int i=0; i<size; i++)
       philosophes[i]=new Philosophe(i,size,fourchettes).start();
  }
}
```