Théorie et pratique de la concurrence – Master 1 Informatique
TD 3 : Sémaphores

Exercice 1 : Nous considérons l'exemple ci-dessous

```
semaphore S <- 1, T <-0
```

```
process P
p1: wait(S)
p2: write("p")
p3: signal(T)
```

```
process Q
q1: wait(T)
q2: write("q")
q3: signal(S)
```

1. Quelles sont les sorties possibles de cet algorithme ?
2. Si l'on efface l'instruction `wait(S)`, quelles sont les sorties possibles ?
3. Et si l'on efface, l'instruction `wait(T)` ?

Exercice 2 : On considère l'algorithme suivant pour le dîner des philosophes (avec 5 philosophes). On suppose que les opérations d'addition sont faites `modulo 5`.

```
semaphore array[0..4] fork <- [1,1,1,1,1]
semaphore room <- 4
```

```
process Philo i
loop forever
p1: think
p2: wait(room)
p3: wait(fork[i])
p4: wait(fork[i+1])
p5: eat
p6: signal(fork[i])
p7: signal(fork[i+1])
p8: signal(room)
```

1. Montrez que à tout instant chaque fourchette est prise par au plus un philosophe.
2. Montrez que en considérant des sémaphores faibles (pour lesquels lors de l'envoie d'un `signal` au moins un des processus en attente prendra la main) cet algorithme respecte la propriété d'absence de famine.
3. Que se passe-t-il si on considère que `room` est initialisé à 3 au lien de 4 ?
4. Dans ce dernier cas, il y a-t-il une solution pour garantir l'absence de famine ?

Exercice 3 : Dans une ville tranquille, un coiffeur possède un petit salon ayant une porte d'entrée, une porte de sortie, un fauteuil de coiffure et N chaises. Les clients arrivent par la

porte d'entrée et sortent par la porte de sortie après avoir eu leur coupe de cheveux. Comme le salon est petit, uniquement le client sur le fauteuil de coiffure peut-être servi à un moment donné par le coiffeur. Le coiffeur passe sa vie entre dormir et servir ses clients. Quand il n'a aucun client, le coiffeur dort. Quand un client arrive et le fauteuil est libre, il s'assoit dans le fauteuil et il réveille le coiffeur. Si le fauteuil n'est pas libre, le client occupe une chaise s'il y a des chaises de libre ou il attend qu'une chaise se libère sinon. Un client sur une chaise attend que le fauteuil se libère. Après avoir fini une coupe, le coiffeur fait sortir le client servi et s'endort.

Modélisez ce problème en utilisant des sémaphores pour la synchronisation entre le coiffeur et ses clients. Pour cela écrire dans le langage similaire à celui vu en cours :

1. un processus `Client` dans lequel sont identifiés clairement ses états : en attente d'une chaise, sur une chaise en attente du fauteuil, sur le fauteuil en attente de la fin de sa coupe et servi,
2. un processus `Coiffeur` avec les états endormi et en service
3. les déclarations de sémaphore et leur valeur initiale
4. exprimer en LTL la propriété suivante : "Le coiffeur est réveillé que si le fauteuil est occupé"

Exercice 4 : *Algorithme de Barz*

Nous considérons l'algorithme suivant qui permet à au plus $k > 0$ processus d'accéder en section critique.

Algorithme de Barz

```
semaphore S <- 1
semaphore gate <- 1
int count <- k
```

```
process Philo i
loop forever
     non-critical section
p1:  wait(gate)
p2:  wait(S)
p3:  count <- count - 1
p4:  if count > 0 then
p5:    signal(gate)
p6:  signal(S)
     critical section
p7:  wait(S)
p8:  count <- count + 1
p9:  if count = 1 then
p10:   signal(gate)
p11: signal(S)
```

1. Expliquez pourquoi on peut réécrire cet algorithme peut être écrit sous la forme simplifiée donnée à la page suivante.

Algorithme de Barz simplifié

```
semaphore S <- 1
semaphore gate <- 1
int count <- k
```

```
process Philo i
loop forever
     non-critical section
q1:  wait(gate)
q2:  atomic{
       count <- count - 1
       if count > 0 then
         signal(gate)
     }
     critical section
q3:  atomic{
       count <- count + 1
       if count = 1 then
         signal(gate)
     }
```

2. On dénote par *entering* la $q2$ et $inCS$ est $q3$. On dénote par $\#entering$ [respectivement $\#inCS$] le nombre de processus pour lesquels *entering* [respectivement $inCS$] est vraie à un instant donné. Montrez que la conjonction des formules suivantes est un invariant :
   (a) $entering \Rightarrow (gate = 0)$
   (b) $entering \Rightarrow (count > 0)$
   (c) $\#entering \leq 1$
   (d) $((gate = 0) \wedge \neg entering) \Rightarrow (count = 0)$
   (e) $(count \leq 0) \Rightarrow (gate = 0)$
   (f) $(gate = 0) \vee (gate = 1)$
3. Montrez que la formule $count = k - \#inCS$ est un invariant.
4. En déduire que à tout moment le nombre de processus en section critique est inférieur ou égal à k.