Programmation Réseaux et Concurrente
Master 1 Informatique UPEC 2015/2016
TP 4 : Exclusion Mutuelle

Exercices

Exercice 1:

Impléméntez l'algorithme d'exclusion mutuelle suivant

```
int turn := 1 // variable partagée

-- Thread T
loop forever :
T1: section NC
T2: await turn==1
T3: section critique
T4: turn:=2

-- Thread S
loop forever :
S1: section NC
S2: await turn==2
S3: section critique
S4: turn:=1
```

Trouvez le moyen d'observer la famine.

Exercice 2:

Implémentez l'algorithme de Dekker

```
boolean D1 := False
boolean D2 := False
int turn := 1

-- Thread T
loop forever :
T1: section NC
T2: D1 := True
T3: while (D2 == True) :
T4:     if (turn == 2)
T5:           D1 := False
T6:           await (turn == 1)
T7:           D1 := True
T8: section critique
T9: turn := 2
T10: D1:=False
```

```
-- Thread S
loop forever :
S1: section NC
S2: D2 := True
S3: while (D1 == True) :
S4:     if (turn == 1)
S5:           D2 := False
S6:           await (turn == 2)
S7:           D2 := True
S8: section critique
s9: turn := 1
S10: D2:=False
```

Observer le comportement de votre programme.

Exercice 3:

Implémentez l'algorithme de Peterson.

Exercice 4: (***)

Exclusion mutuelle entre n threads : le tournoi (Peterson et Fischer, 1977). On a n threads où pour simplicité on suppose $n = 2^k$. $T0, T1, \ldots$ Les threads se font face deux à deux dans un tournoi à k tours. Le gagnant de chaque match se décide avec un des algorithmes pour deux threads (Dekker, Peterson). Dans chaque tour t (entre 0 et $k-1$) il y a 2^{k-t-1} matchs à faire, chaque match est donc identifié par le nombre de tour t, et un deuxième nombre i entre 0 et 2^{k-t-1}. Pour chaque match on va ici utiliser l'algorithme de Peterson et on doit donc disposer de l'equivalent des variables partagées turn, D1 et D2, pour cela on utilise des tableaux. Pour le match t, i on aura les variables `turn[t][i], D[t][2i], D[t][2i+1]`. Chaque thread doit savoir quel match il joue et pour cela il a des variables locales : `t,i`. De plus il doit connaître son rôle dans l'algorithme de Peterson, et pour cela il a une variable `id` qui prend les valeurs 1,2.

Pour le thread numéro `j` on a donc l'algorithme :

```
loop forever:
1  Section NC
2  i:= j
3  for t:= 0 à k-1 faire
4     id := (i%2) + 1
5     i := i div 2
6     D[t][2i + (id-1)] := true
7     turn[t][i] := 3 - id
8     await(not D[t][2i + 2 - id] || turn[t][i] == id)
9  Section critique
10 for t = k - 1 à 0 faire
11    D[t][2i] := false
```

Implèmentez l'algorithme de Peterson et Fischer pour k=4.