Université Paris 7
Master 1 II – Théorie et pratique de la concurrence
24 mai 2005

Durée : 3 heures.
Documents autorisés : notes de cours et TD. Le barème est indicatif.

Question 1 (3 points)

Soit le programme S suivant :

```
parbegin
     atomic { (x >= 3) -> x = x-3 }
||   atomic { (x >= 2) -> x = x-2 }
||   atomic { (x == 1) -> x = x+5 }
parend
```

1. Quelles sont les valeurs initiales de `x` pour lesquelles le programme termine dans l'hypothèse d'une politique d'ordonnancement avec équité faible ? Justifiez votre réponse.
2. Pour les valeurs initiales trouvées, quelles sont les valeurs finales de `x` ? Justifiez votre réponse.
3. Développer une preuve pour l'assertion $\{x > 0\} \;\; S \;\; \{x > 0\}$.

Question 2 (2 points)

Soit le programme concurrent suivant :

```
int x = 0;
sem s1 = 1, s2 = 0;
parbegin
     P(s2); P(s1); x= x*2; V(s1)
||   P(s1); x = x*x; V(s1)
||   P(s1); x = x+3; V(s2); V(s1)
parend
```

Indiquer quelles sont les valeurs finales de la variable `x`. Expliquer votre réponse.

Question 3 (5 points)

Supposons qu'une machine fournit l'instruction atomique suivante :

```
flip(int lock) :
     atomic { lock = (lock+1) % 2; /* inverse le lock */
              return (lock) }      /* renvoie sa nouvelle valeur */
```

Un programmeur propose la solution suivante pour l'implementation d'une section critique pour deux processus :

```
int lock = 0; /* variable partagee */

proctype CS(int i) /* i est soit 1 soit 2 */
{   do :: true ->
           do :: (flip(lock) != 1) ->
                    do :: lock != 0 -> skip
                       :: else -> break
                    od
              :: else -> break
           od
           /* section critique */
           lock = 0;
           /* section non critique */
    od
}
```

1. Expliquer-lui pourquoi sa solution ne marche pas en lui montrant une trace d'exécution dans laquelle les deux processus se retrouvent en même temps dans leur section critique.
2. Il vous propose de garder sa solution mais de changer dans la machine l'instruction atomique `flip` : au lieu d'une addition modulo 2, utiliser une addition modulo 3.

   Cette nouvelle solution est-elle correcte pour deux processus ?

   Justifier votre réponse (si la réponse est positive donner un invariant, sinon exhiber une trace d'exécution).
3. Pour cette dernière solution, commenter informellement les propriétés d'absence de blocage, de délai et de famine.

Question 4 (5 points)

Supposons que des atomes de hydrogène et d'oxygène flottent dans l'air en essayant de se combiner ensemble en molécules d'eau. Pour cela, il faut que deux atomes de hydrogène et un atome d'oxygène se synchronisent ensemble. On simule les atomes de hydrogène (H) et oxygène (O) par deux processus. Chaque processus H appelle une procédure `Hready` quand il désire se combiner dans une molécule d'eau. Chaque processus O appelle une autre procédure `Oready` quand il désire se combiner.

Un atome H doit être bloqué dans la procédure `Hready` jusque un autre atome H appelle `Hready` **et** un autre atome O a appelé `Oready`. Ensuite, l'atome O du triplet doit appeler une procédure `makeWater`. Quand `makeWater` termine, les trois atomes qui se sont ainsi synchronisés doivent retourner de leurs appels de `Hready` et `Oready`. Plusieurs molécules d'eau peuvent se former ainsi en même temps.

Écrivez les trois procédures en utilisant les sémaphores pour la synchronisation. En plus des opérations `P` et `V` classiques sur les sémaphores, vous pouvez utiliser une opération `tryP` qui est la variante non-bloquante de l'opération `P` : elle effectue `P` si possible et renvoie vrai, sinon elle continue et renvoie faux.

Question 5 (6 points)

Soit le problème du dîner des philosophes vu en cours.

1. Écrire un moniteur pour résoudre ce problème de synchronisation. Le moniteur **doit** avoir deux procédures : `getforks(id)` et `relforks(id)`, où `id` est l'identité du philosophe qui appelle. Utilisez le langage algorithmique du cours avec la politique "signale et continue". Cette solution ne doit pas se soucier de l'équité. N'oubliez pas de spécifier l'invariant du moniteur et la sémantique de chaque variable conditionnelle !
2. Donner une implémentation en Java de votre moniteur. Expliquer comment vous avez implémenté les variables conditionnelles utilisées ci-dessus.
3. Modifier la solution algorithmique pour obtenir une solution équitable. Expliquer votre solution.