M2 FIIL Software Model Checking
Année 2019-2020

TP2 sur Cubicle

Le but de ce TP est de modéliser et de prouver des programmes concurrents en utilisant Cubicle (`http://cubicle.lri.fr`) et son extension aux mémoires faibles (`http://cubicle.lri.fr/cubiclew`).

1 Algorithme de Peterson

Nous partons de l'implémentation de l'algrithme de Peterson suivante où `Turn`, `Want` et `Other` sont des variables globales :

```
void init () {
  Turn = P1;
  Want[P1] = False; Want[P2] = False;
  Other[P1] = P2; Other[P2] = P1;
}

void process (int i) {
  while (True) {
  L1:
    Want[i] = True;
  L2:
    Turn = Other[i];
  L3:
    while ( Want[Other[i]] && !(Turn = i) ) { ; }
  L4_crit:
    // section critique
  L5:
    Want[i] = False;
  L6:
    // section restante
  }
}
```

La modélisation de cet algorithme en Cubicle est donnée dans le fichier `peterson.cub` disponible dans l'archive `http://www.lri.fr/~conchon/FIIL/tp-cubicle.tgz`.

Question 1. En utilisant Cubicle, prouver que tel qu'il est modélisé, l'algorithme de Peterson est sûr (il ne peut pas y avoir deux processus simultanément en section critique) et que l'arrêt d'un processus en dehors d'une section critique n'empêche pas l'autre d'y entrer.

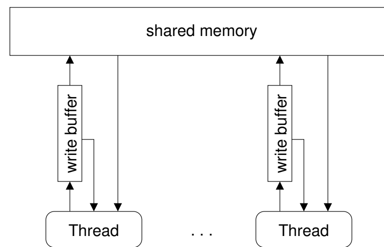
Question 2. Modifier l'algorithme pour introduire des problèmes de sûreté et d'accès à la section critique. Donner des traces qui exhibent ces problèmes.

Question 3. Prouver que tel qu'il est modélisé, l'algorithme de Peterson n'a pas de deadlock.

Dans le fichier `peterson.c`, l'algorithme de Peterson a été utilisé pour garantir l'exclusion mutuelle de deux threads qui incrémentent 1 000 000 de fois chacune une variable globale `cpt`.

Question 4. Tester le programme `peterson.c` sur votre machine et vérifier qu'il affiche ≪ `cpt = 2000000` ≫.

Les machines multi-coeurs se comportent comme si les écritures en mémoire se faisaient à travers un buffer. Ainsi, une affectation revient à écrire une valeur dans un buffer et cette valeur est copiée de façon asynchrone en mémoire centrale.



Sur ces machines, la sémantique par entrelacements des threads n'est pas respectée. Par exemple, le programme suivant peut terminer avec `r1 = 0` et `r2 = 0` si initialement `x = 0` et `y = 0` :

Thread 0	Thread 1
`x := 1;` `r1 := y;`	`y := 1;` `r2 := x;`

Question 5. En utilisant `cubiclew`, la version pour mémoire faible de `cubicle`, modifier le fichier `peterson.cub` pour modéliser l'exécution du programme sur une machine multi-coeur avec mémoire faible. Donner une trace qui conduit à un problème de sûreté.

Il existe une instruction `fence` dont l'effet correspond à vider le buffer local au thread qui l'exécute.

Question 6. Toujours en utilisant `cubiclew`, déterminer où utiliser cette instruction pour rendre le code de l'algorithme Peterson de nouveau correct.

Question 7. En utilisant la macro suivante, corriger le fichier de `peterson.c`.

```
#define fence()  __asm__ __volatile__ ("mfence":::"memory")
```